AFIPS

CONFERENCE PROCEEDINGS

VOLUME 40

1972

SPRING JOINT COMPUTER CONFERENCE

May 16 - 18, 1972
Atlantic City, New Jersey

AFIPS PRESS
210 SUMMIT AVENUE
MONTVALE, NEW JERSEY 07645

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1972 Spring Joint Computer Conference Committee or the American Federation of Information Processing Societies, Inc.

Library of Congress Catalog Card Number 55-44701
AFIPS PRESS
210 Summit Avenue
Montvale, New Jersey 07645

Printed in the United States of America

CONTENTS

IMPLEMENTATION OF PROGRAMMING LANGUAGE PROCESSORS

FRONTIERS OF LARGE-SCALE SCIENTIFIC COMPUTATION

THE COMPUTER AS SUPERTOY—PANEL SESSION

(No papers in this volume)

TRAINING COMMERCIAL PROGRAMMERS AND SYSTEMS ANALYSTS

SOFTWARE DESIGN FOR THE MINI-COMPUTER

TECHNIQUES FOR DEVELOPING LARGE PROGRAMMING SYSTEMS

MATHEMATICAL OPTIMIZATION THEORY AND ALGORITHMS

NEW THEORETICAL FRONTIERS OF COMPUTER SCIENCE

THE ARPA NETWORK

PROGRAMMING LANGUAGES FOR SPECIALIZED APPLICATION AREAS

NEW TRENDS IN THE ARCHITECTURE OF COMPUTER SYSTEMS

COMPUTERS IN INSTRUCTION—SOME CONSIDERATIONS

OPERATING SYSTEM DESIGN CONCEPTS

GRAPHIC TERMINALS—PRESENT AND NEXT STATES

FORMAL ANALYSIS OF ALGORITHMS

THE COMPUTER IN GOVERNMENT—A TOOL FOR CHANGE

INTERACTIVE SYSTEMS

DATA COMMUNICATIONS—THE PAST FIVE YEARS AND THE NEXT TEN YEARS

MANPOWER FOR COMPUTERS—HEYDAY OR CUTBACK

MICROPROGRAMMING ENTERS A NEW ERA

PERFORMANCE PREDICTION—MODELING AND MEASUREMENT

LSI PERSPECTIVES—ARCHITECTURE AND COST OF SMALL COMPUTERS

COMPUTER SIMULATION AS A DECISION MAKER'S TOOL

THE DILEMMA OF INSTALLATION MANAGEMENT

PROGRAM DOCUMENTATION—PANEL SESSION

(No papers in this volume)

THE ROLE AND SCOPE OF COMPUTER SYSTEMS DESIGN RESEARCH IN A UNIVERSITY SETTING—PANEL SESSION

(No papers in this volume)

AN EVALUATION OF THE STATE OF COMPUTER SCIENCE EDUCATION

SCIENTIFIC COMPUTATION—THE SOCIAL SCIENCES

PROGRAMMING FOR PROCESS CONTROL AND REAL TIME APPLICATIONS

ACM PRIZE PAPERS IN PROGRAMMING LANGUAGES AND SYSTEMS

A SPECTRUM OF MEMORY STORAGE SYSTEMS, NOW AND IN THE FUTURE

GRAPHIC SOFTWARE

COMPUTERS IN SECONDARY EDUCATION

LSI PERSPECTIVES—DESIGN AUTOMATION: DESIGN AND SIMULATION

DEVELOPMENTS IN BIOMEDICAL COMPUTER TECHNOLOGY

THE IMPLEMENTATION GAP IN MANAGEMENT INFORMATION SYSTEMS

LSI PERSPECTIVES—THE BOUNDARIES OF COMPUTER PERFORMANCE

An appraisal of compiler technology

by ROBERT M. McCLURE

Consultant

INTRODUCTION

Although the last decade has seen the implementation of a very large number of compilers for every conceivable language, the literature on compiler writing is still unorganized. There are many papers on formal languages, syntactic analysis, graph theoretic register assignment, and similar topics to be sure. But, they are not very useful in helping the newcomer to the field decide how to design a compiler. Even the recent books in the field are more encyclopedic in nature than instructive. The best single book available is Gries,[1] which has a good bibliography for further study.

The few open descriptions of compilers that do exist rarely are candid about the mistakes that were made. This is, after all, human nature. Moreover the nature of scientific publishing is not conducive to papers of an evaluative or subjective nature. The principal reason, therefore, for writing this paper, is not to add to the basic body of scientific knowledge, but rather to provide a few value judgments on how compilers are being written and should be written. Since subjective statements should always be prefaced with "It is my opinion that", the indulgent reader will understand that this phrase should be applied to the entire paper.

There is an enormous amount of material to be studied in connection with compiler design. Most of it is very difficult to read. We refer to the listings of the compilers themselves and associated internal documentation. At present there is no alternative to obtaining a comprehensive knowledge of the field. Even if one is willing to put forward the effort, however, much of the material is difficult to obtain. The desire of computer manufacturers and software houses to protect their latest ideas interferes with a free flow of information. The result of this is the growth of an oral tradition for passing information much like the wandering troubadours of yore. Until this changes, there will be hardships worked on those who would like to learn the trade.

In order to reduce this paper to manageable size and to make generalizations more useful, let it be understood that we are mainly talking about so-called "production" compilers. By this we mean compilers for important languages that are intended for translating programs that are expected to be run extensively. Moreover, we restrict our consideration to compilers for medium and large scale general purpose computers. The subject of compilers for minicomputers and special purpose computers is deserving of considerable study on its own. Also research compilers, "quick and dirty" compilers, and pedagogic compilers will not be considered in this commentary. Finally, it is inevitable that someone's favorite technique will be given a short straw. For this we plead for tolerance.

SYNTACTIC ANALYSIS

In the area of syntactic analysis or parsing, the necessary solutions are clearly at hand. Parsers are now routinely constructed by graduate students as class projects. The literature on the subject is extensive, and new techniques are revealed regularly. Almost all compilers are written using only two of the many available methods however: recursive descent or precedence. Rarely are either used in pure form. Compilers that use both methods intermixed are common. Although both of these methods have their roots in the very earliest compilers, they remain the mainstays of the business.

Recursive descent

The basic idea in recursive descent is that there is a one-to-one correspondence between the syntactic units in the formal description of the language and routines for their recognition. Each routine examines the input text by either scanning for specified terminal tokens (usually produced by a separate lexical analyzer) or by calling other routines to recognize non-terminal syntactic units. Figure 1 is a flowchart of one such simple recognizer. Because most practical languages are defined recursively, the recognizers must themselves be

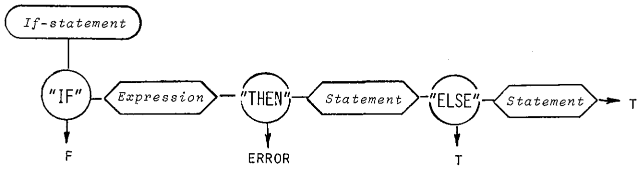


Figure 1—Recognition routine

recursive. Upon exit from each procedure, either the required syntactic unit has been recognized and the appropriate actions taken, or an error message has been produced.

Recursive descent analyzers are usually written so that it is never necessary to "back up" the input stream. If an impasse is reached, an error indication is given, perhaps a correction attempted, and the scan resumed at some readily recognizable token, such as ";". The knowledge of the context at the time an error is discovered allows more meaningful diagnostics to be generated than with any other technique.

The use of recursive descent usually requires the rearrangement of the formal syntax, since left recursions must be removed. Although this is not difficult, it does somewhat spoil the clarity of the approach. For peculiar grammars, it is easier to use recursive descent than to transform the grammar into a form suitable for precedence analysis. Although the methodology is basically ad hoc, recursive descent is the most widely used method of syntactic analysis.

Recursive descent has one further advantage for a language with a large number of productions. Precedence methods seem to require table sizes proportional to the square of the number of syntactic units, whereas recursive descent recognizers are roughly proportional to the grammar size.

Precedence analysis

The idea that operators have varying degrees of "precedence" or "binding-power" is quite old, and has been used in some of the earliest compilers. The formalization of precedence relations between operators actually started with Floyd[2] in 1963. Now it is more customary to define precedence relations between all syntactic units. While very few production compilers have used a formal precedence parser yet, the modern implementation of these techniques such as described by McKeeman,[3] is clearly destined for wider use.

The fundamental attraction of precedence methods lies in the automatic construction of analyzers from the formal grammar for the language to be analyzed. Parsers built in this way can be quickly changed to reflect language changes simply by changing the tables without modifying the underlying interpretation algorithm. Furthermore, if the tables are constructed by formal methods, the language accepted is exactly that specified and no other. This considerably simplifies the construction of bug-free parsers.

There are some drawbacks to parsers built wholly around precedence methods. For example, many languages in everyday use, such as COBOL and FORTRAN can simply not be made into precedence languages. Even PL/I can only be fit into the correct mold with considerable difficulty. Moreover, the tables required can be quite large.

A major problem with precedence parsing methods is the problem of recovery from source program errors and the issuance of good diagnostics for such errors. Several approaches to solving this problem have been tried with only modest success. Various techniques for reducing the size of the tables required, such as the introduction of precedence functions, serve only to complicate this problem.

Nevertheless, there are several conditions that suggest strongly that a precedence parser of some variety should be tried. If, for example, a parser must be produced in the shortest possible time. Precedence parsers tend to be easy to debug. A further advantage is gained if the language is being designed at the same time as the compiler, since the language can be made into a precedence language. Finally, a precedence parser is often a most suitable alternative when recursive procedure calls required for recursive descent parsing are either impossible (as in FORTRAN) or expensive (as in PL/I).

Mixed strategies

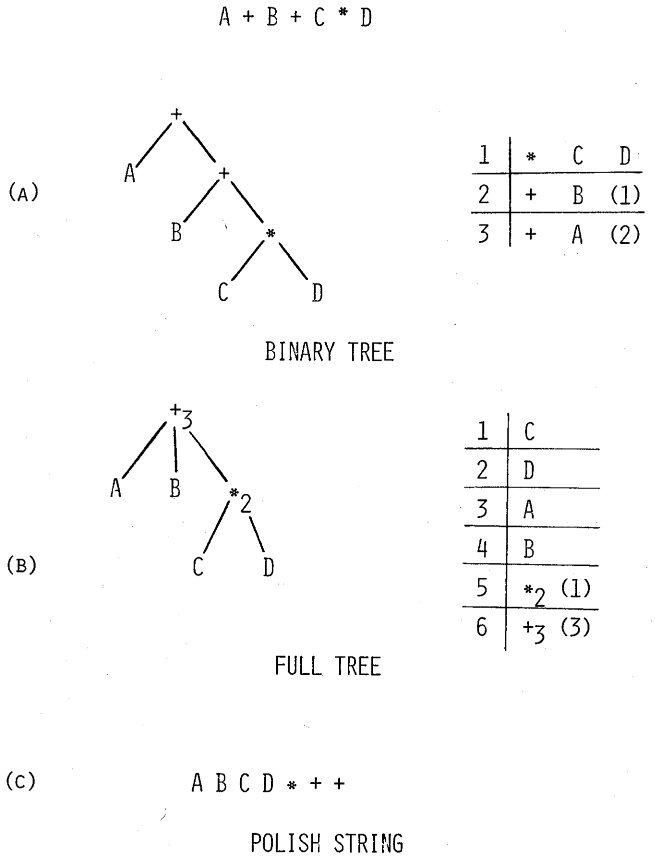
A number of compilers use the technique of recursive descent for analysis of the larger syntactic units such as statements, and turn to precedence methods for parsing expressions. In this case, simple operator precedence analysis is customarily used. There is no conceptual or practical difficulty in accomplishing this since the expression recognizer requires relatively small tables which may easily be hand constructed. The number of calls on procedures is minimized (normally a high overhead item) and perhaps the best of both worlds is achieved.

The idea of mixing parsing techniques can be generalized. It appears (with no theoretical foundation as yet) that a good partition is achieved when the subgrammar to be parsed with a precedence parser results in a relatively dense precedence matrix, and the subgrammar to be parsed with a recursive descent parser would result in a relatively sparse (for legitimate symbol pairs) precedence matrix.

INTERNAL REPRESENTATION

Generation of code occurs simultaneously with syntactic analysis only in very small compilers and quick and dirty compilers. In most compilers the results of syntactic analysis are stored for later code generation. A number of ways have been described for the internal representation of the results of parsing, two of which have attained really wide usage: tree structures (most usually a form known as triples) and late operator Polish notation. Figure 2 illustrates several forms of the representation of a simple expression. Form (a) is a binary tree, called triples when recorded in the tabular form shown, since each line consists of an operator and two operands. The result of each operator is known implicitly by the line number of the triple. Form (b) is a tree form with variable sized nodes. Form (c) is the Polish form and is actually the string that results from traversing the terminal nodes of a parse tree in a prescribed order.

The simpler compilers prefer the Polish representation since subsequent processing usually amounts to simply a linear reading of the text. Fortunately the order in which action is to be taken by the code generator is very nearly that of tokens in the Polish string anyway. Since this is also the order of generating the Polish string, the only advantage gained from deferred generation is that obtained by a full reading of all of the text, including declarations, implicit as well as explicit. Although optimization can be implemented on internal representations in Polish form, tree forms are much easier to work with.



1	*	C	D
2	+	B	(1)
3	+	A	(2)

1	C
2	D
3	A
4	B
5	$*_2$ (1)
6	$+_3$ (3)

Figure 2—Internal representations

A further advantage of Polish strings is that they may be easily written on and read from sequential data files, are conceptually simple, and require a minimum of additional space for linkage information. If memory space is at a premium or the simplest representation is preferred, Polish strings are recommended.

Complete representation of the source program in tree form is now growing in popularity, and has appeared in quite a number of the more recent compilers. It is especially prevalent in compilers for major languages for the larger machine in which optimization is important. The ease with which the program can be searched and/or rearranged is the primary motivation for this selection. For building a generation system based upon formal semantics, an internal tree representation is a good choice. Not only does it appear that code generation can be formalized but also that optimization strategies can be formalized utilizing the idea of transformation sets defined over trees.

SYMBOL TABLES AND DICTIONARIES

Considerable effort has gone into devising symbol table routines that have all of the desirable properties of compactness of storage required, speed of access, and simplicity of construction. The result is that almost all compilers use some variant of the hash link method. In this method, the symbol to be looked up is first hashed into a reasonably small range such as 31 to 512. This hash index is then used to access the head of a list of symbols with the same hash value. This is shown schematically in Figure 3. This hash chain is frequently ordered, say alphabetically, to reduce further the lookup time.

The dictionary information associated with each symbol may either be included directly with the external representation or be contained in a separate table. Since in a multiple pass compiler, the external representation is not required after the lexical analysis is complete, the separation of symbol table and dictionary has become customary.

It is becoming increasingly important to make provision for symbol tables and dictionaries to grow beyond the bounds of the available main memory. This presents no unusual problems for hardware systems that

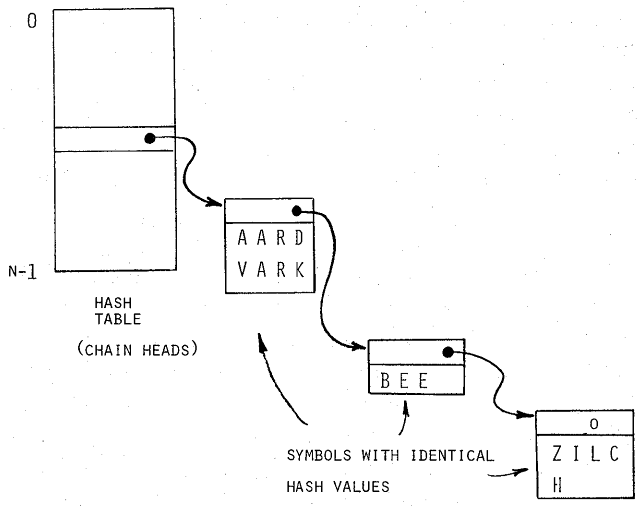


Figure 3—Hash chain symbol table

support virtual memory, but does require careful attention if it must be strictly software supported. The simplest and usual method is to divide the space required into pages and to address this space through an availability table. This method is used in the IBM PL/I compiler (F-Level) to allow the compilation of very large programs on a relatively small machine. In designing symbol table methods for use with software virtual memory, a premium is placed on minimizing the number of references to different blocks.

Compilers for block structured languages with immediate code generation usually allocate new dictionary entries in stack fashion. That is, new entries are placed in first position on the correct hash chain. Searching the hash chain then automatically gives the correct instance of the symbol searched for. At block closure, all the chains are pruned until an enclosing block number is found. The table is then shortened to the level that existed at block entry. In this way, symbol table entries that are no longer needed by the compilation are constantly discarded. Although PL/I is block structured, this very simple approach is not available without refinement. In PL/I, declarations may at any point be made in enclosing blocks. For examples, lables on entry statements belong to the immediately enclosing block, implicitly declared variables belong to the outermost block, and variables declared with the EXTERNAL attribute are scopeless.

TABLE MANAGEMENT

Early compilers had fixed size tables for storing information needed during compilation. Since the relative size of these tables could only be statistically determined, unnecessary limitation in the size of programs that could be compiled frequently occurred. During the 1960's however, dynamic allocation of storage came into widespread use. Although many techniques have been invented, only two have proven extremely popular: movable, contiguous tables, and block allocation.

Movable tables

The conceptual simplicity of having tables in consecutive memory locations is a major reason for adopting the idea of "floating" tables. The additional burden of referencing all entries indirectly through a current table base pointer is a small price to pay for this simplicity. First used in compilers by Digitek in a series of minicomputer compilers and by IBM in a COBOL compiler for the IBM 7090 in the early 1960's, the basic ideas have been widely adopted. Modern computer architecture that makes both base addressing and indexing simultaneously available makes implementation extremely simple.

Every table is allowed to grow (or shrink) independently. Before any table expands, a check is made to see if there is available space. If not, space is made available

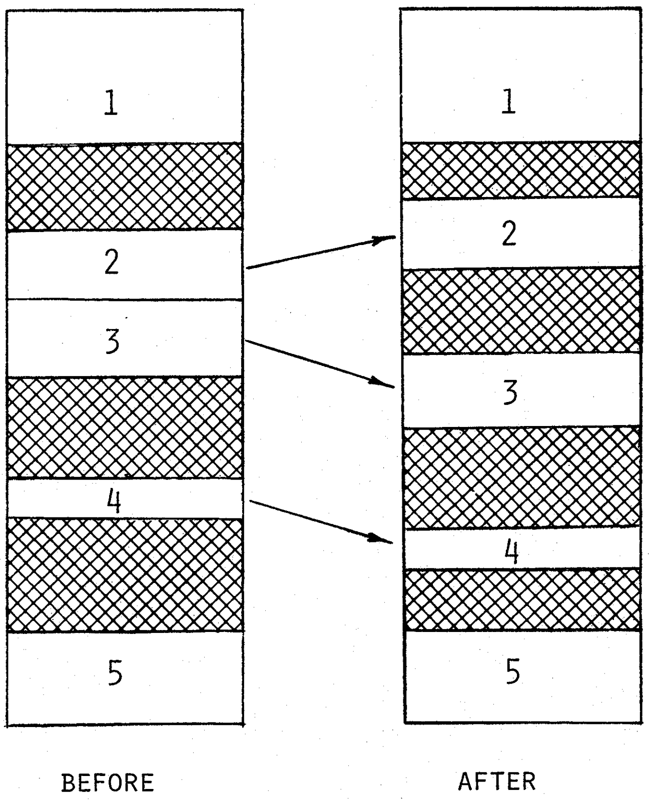


Figure 4—Movable table storage allocation

by actually copying the data items in the tables to be moved into another location in memory. Although copying data in memory to reorganize storage seems inefficient at first, it turns out to be quite satisfactory in practice since it occurs rarely. Figure 4 shows how tables are rearranged to allow available space for all tables to grow.

The interesting questions about this form of allocation revolve around deciding how much of the available space to allocate to each table. Garwick suggests that space be allocated in proportion to how much each table has grown since the last allocation. CITRUS (the storage allocation package in the IBM COBOL compiler) required specification of the desired increment of space. Most common is dividing the remaining space proportional to the current size of each table (with some prescribed minimum). All methods will run out of space only when there is absolutely no more. This may not be desirable, though, since considerable time will be spent trying to squeeze in the last item before overflow occurs.

Movable tables seem to work best for relatively small memories, for machines in which internal speeds are fast relative to I-O speeds, or for systems in which all available memory is allocated to the compiler at the start of any compilation.

List structures are complicated by the necessity to use relative links rather than absolute addresses and to identify the table pointed to.

Block allocation

Block allocation methods are the second most popular storage management technique. In this case, tables are not kept in contiguous locations, and information is usually linked together. Each routine that may add data to a table is responsible for requesting a block of suitable size (sometimes a system fixed size) for the purpose at hand. Usually space is allocated in multiples of the basic element size of the table at hand in order to avoid calling the main allocator too often. Since OS/360 implements this form of allocation as a primitive in the operating system, this form of allocation has been quite widely used on that machine.

This technique does have the principal advantage that since table segments are not moved after allocation, absolute pointers may be used in list structures. It suffers from the drawback that storage fragmentation can occur and may prevent usage of all available memory.

Block allocation is suggested whenever memory is a resource to be shared with other simultaneous users of the machine in a multiprogramming environment. This is because it is desirable to minimize the amount of memory required at any given time, and most main memory allocation algorithms supported by operating systems do not guarantee that additional memory will be allocated contiguously with previously allocated memory.

IMPLEMENTATION TECHNIQUES

Early compilers were invariably written in assembly language; most still are today. It was originally felt that only assembly language could yield the efficiency that was required in such a widely used program as a compiler, and that the required operations were not well supported by higher level languages. Although it has now been generally recognized that very little of any compiler needs to be "tightly coded" to achieve nearly the same efficiency, the tradition of assembly coding is dying a slow and painful death.

A second reason usually given for writing a compiler in assembly language was to minimize the space that the compiler required for its code. Factors of 1.5 to 3 have been cited. With the growth of main memory sizes available, the almost universal availability of random access secondary storage, and the common use of dynamic loading techniques, the requirement for small code has been considerably reduced in importance.

Advocates of coding in "higher level" languages have not always been completely candid in their arguments either. It is frequently stated that one of the main motives for using a higher level language is the gain in readability that occurs. Anyone who has tried to read a compiler written in FORTRAN knows that this simply is not the case. A much stronger case may be made for PL/I or ALGOL. The fluidity of these languages plus the natural benefits of block structuring generally result in code substantially more readable than assembly code.

A major drawback to most higher level languages for coding compilers is that the native data types manipulated in these languages are neither those required in either lexical scanning, nor those required for table management. Both of these are vitally important in compiling, and result in the construction of subroutines, frequently in assembly code, to support them. The linkage overhead in using these routines can be substantial.

By all odds, the most compelling reason for using a higher level language is the conciseness with which code can be written. The sheer laboriousness of assembly code is a major obstacle to its use. One of the best measures of the suitability of a language for a particular purpose is the number of characters that must be written to achieve the desired result. Since error rates in programming are almost independent of the language being written, concise programs will have fewer bugs than verbose programs.

Pops

One approach that has gained a number of particularly ardent adherents is that of writing compilers in an interpretive language. This idea seems also to have originated in the early 1960's. Although the COBOL compiler for the IBM 1410 was written in interpreted code, the main source of the popularity was the series of (mostly FORTRAN) compilers written by Digitek. A number of syntax directed compilers of the same vintage utilized an internal representation of a similar nature. The increased suitability of current computer instruction sets for compiler writing has caused the technique to largely fall from favor for large machine compilers in recent years. The technique has much to recommend it for some applications, though, and it is worthy of a few comments.

Since there have been no published papers on the Digitek POP system, which appears to be the most highly developed system, we will include here a somewhat more complete description than for the other ideas discussed in this paper.

The name POP derives from the *P*rogrammed *OP*erators on the SDS 910 for which the first Digitek FORTRAN compiler was written. These were a series of unassigned operation codes that caused a trap to a location distinct for each such op-code. This enabled the user to define the meaning of each of these op-codes by a subroutine. Subsequently, for other machines which did not have such a feature, a completely interpretive system was substituted.

The POP system consists of a set of operations that resemble single address instructions. The single operand in each POP is either a single data element (usually an integer but perhaps a pointer into a table, etc.), a character, a string of characters (this is handled indirectly), a table (movable), or a flag. Additional data types are defined as needed. Tables are implemented as contiguous, movable tables as previously described. A pointer is defined as having a table number part plus an offset into a table. Pointers are used for forming linked structures and for referencing data in tables if other than the last item is to be referenced. Tables are normally used like stacks. One table is distinguished as the main stack and is often used as an implicit second operand. Another table is distinguished as the stack used to save return addresses in procedure calls. Recursion is therefore quite natural.

To illustrate how this system is used, we will define several of the more common POP's. The first POP is LDS (Load Stack), and is written as:

```
LDS    A
LDS    B
```

This sequence of two POP's is interpreted as follows: First the data item A, a single work item, perhaps an integer, is placed on the main stack, extending it by one word in the process. Then the item B is added to the stack, extending it once again. At the conclusion, the stack appears as in Figure 5. The end of stack item is referred to as item 0 and the next item (A) is referred to as item 1.

The POP STS (Store Stack) is simply the converse. For instance:

```
STS    A
STS    B
```

stores the end item of the stack in cell A, and shortens the stack. The second POP stores the new end item in cell B and shortens it once again. The net effect of the four POP's we executed is to exchange cells A and B.

Similarly, the stack may be loaded with the end item on any table by MOF (Move Off) which has as an operand a table number. The effect of this is to lengthen the stack and to shorten the table specified as the operand of the instruction by moving one word of data. This operation is usually defined to set a global flag to FALSE rather than to move an item if the table specified is empty. The companion operation is MON (Move On). Hence to move the contents of one table into another, the four instruction loop suffices:

```
LA    MOF    TABLE1
      BRT    ALLDONE
      MON    TABLE2
      BRU    LA
```

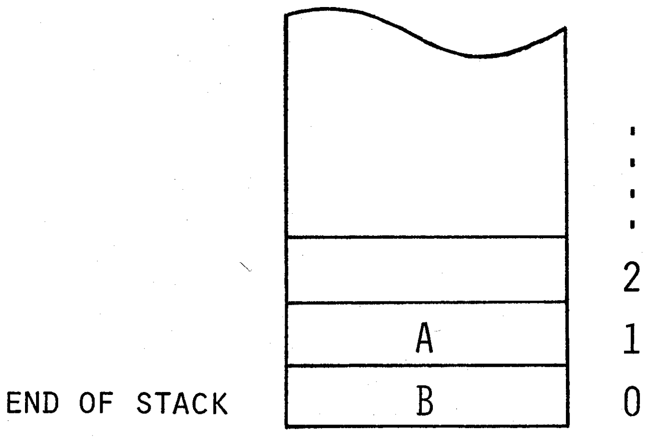


Figure 5—Picture of end of stack

The two new instructions above are BRT (Branch True) and BRU (Branch Unconditional).

Character scanning is done with primitives of the form

```
CHS      character
```

The POP CHS (Character Scan) has as its operand the internal representation of a character. If the specified character is the next in the character input stream, the input scanning pointer is advanced and a global flag set to TRUE. Otherwise the global flag is set to FALSE. Similarly a complete string may be matched with SCN (String Scan), as in

```
SCN      "BEGIN"
```

Subroutine calling is done with BRS (Branch and Save) which stores the return address in a stack. The natural implementation of recursion leads most POP written compilers to be of the recursive descent variety. For instance the sequence of code required to recognize the syntax SUM ::= TERM {+TERM}* is as simple as

```
SUM      BRS      TERM
         CHS      "+"
         BRT      SUM
         RET
```

The POP system is fleshed out with instructions for packing and unpacking data words, creating instructions, setting and testing flags, and so on almost ad infinitum. In theory, a POP compiler can be moved from one machine to another simply by writing a new interpreter for the next machine. In practice, this is not feasible because the bulk of the work in writing a compiler is in designing the code generation strategy, which must be rethought for any new machine in any event.

POP compilers are considerably more compact in code than machine coded compilers, especially where there is an addressability problem (such as in a minicomputer). Fortunately this is the place that compactness is needed most. POP written compilers are, however, slower than machine coded compilers not only because of the interpretive overhead, but also because of the redundant work done in moving data through the stack. This problem is masked in minicomputers since their computing power is very high relative to input output speeds of paper tape and teletypes.

The synopsis is that POPs are a fairly good way to write very small compilers for minicomputers and a poor way to write compilers for large machines.

Translator writing systems

As an implementation technique, the use of one of the many extant Translator Writing Systems deserves at least some mention. The idea of using a TWS is very appealing, but in practice the use of TWS has not proven much of a boon. The reason is quite simple. To date, TWS has done a good job of lexical scan, parsing, and the dictionary parts of a compiler, but has made few inroads into the most difficult parts, namely code generation and optimization. Even if the facilities provided by a TWS are valuable, the penalty of forcing a user into a prescribed mold has been too stiff for most production compiler writers to bear.

CODE GENERATION

Code generation has traditionally been one of the most ad hoc (and bug-ridden) parts of any compiler. There is some evidence, though, that this is changing rather rapidly. Formally, the generation of code is the attaching of semantics to specific syntactic structures. In the case of immediate generation, of course, opportunities for substantially altering the source order are minimal and conversion of the parse back into sequences of instructions proceeds strictly on a local basis. The actual generation of code is accomplished either by open sequences of code that construct the required instructions or by the equivalent of macro sequences selected by combination of the operator and the associated operand types. In the latter case, the macros are usually described in a format similar to the assembly language of the target machine. Conditional features similar to conditional features in macro assemblers are used to improve on the quality of code generated. Provision is normally made to test the state of the target machine as well as supplementary information about the operands. The macros also update the state of the target machine.

In multipass compilers, there is now a move to systematize code generation by formal tree transformations so that all data conversions, register loadings, and the like, are explicitly recognized in the tree structure. Most of the current work in formal semantics is along this line. Information may be collected during tree traversals which aids considerably in the production of high quality code.

Whether the code is produced by conventional or table driven techniques is far less important that the organization of the generation as a sequence of transformations rather than as a complex series of decisions based upon global data left as side effects of prior generation.

OPTIMIZATION

One of the most important differences between production compilers and experimental compilers is that most production compilers try much harder to generate high quality code even at the expense of considerably slower compiling. This tradition dates from the very first FORTRAN compiler. At that time, it was felt necessary to compete with hand generated code in quality in order to gain acceptance. Since that time much effort has gone into finding ways to improve the quality of code generated by compilers. Unfortunately, the matter is too tightly bound up with the specifics of the language at hand. FORTRAN, for example, is relatively easy to optimize. PL/I, on the other hand, is almost impossible due to the significance of side-effects in the language. If, of course, optimization information is supplied by the programmer (which is rarely done), the problem becomes more nearly like that of FORTRAN.

After carefully sifting through all of the forms of optimization that have been used however, there are only two areas of optimization that have sufficient payoff to warrant consideration, unless super-optimization is being attempted. That is, there are only two issues in addition to strictly local improvements in generated sequences. These two areas are register assignment and address calculation.

Optimal register assignment

Also dating back to the earliest compilers is the problem of optimal assignment of the registers available in the machine. If a computer has only one register of a given class, or a very large number, the problem is minimal or does not exist. However, for common machines with 3 to 16 registers in a register class, the advantage to finding the correct data items to maintain in registers can be substantial. This is particularly true of values used for array accessing. Although there have been numerous papers on this subject, the general problem is still unsolved. Even the most satisfactory approaches require an excessive amount of computation to find an optimal solution.

The consequence of this dilemma, is that most compilers that do not do flow analysis usually simplify the problem by merely keeping a simple record of register contents. Loads and stores are then done on a strictly local basis. This works quite satisfactorily for all except the most demanding requirements.

Address calculation

One of the most important forms of non-trivial optimization and one that requires at least a modicum of flow analysis is the calculation of subscripts within a loop by repeated addition of some precalculated value to an address rather than a full recalculation of the address. This is best shown by the simple loop:

```
DO    K = 2 TO 99;
   A(I, J, K) = A(I, J, K +1) + A(I, J, K − 1);
END;
```

The address calculations required can be drastically reduced with only two observations. First, all three array addresses are related to each other with only a constant difference. Second, consecutive executions of the loop body require only that the address be adjusted by a constant amount. The first of these simplifications is called common subexpression elimination. The second is called recursive address calculation. (It has nothing to do with recursion in the programming sense. Here it is really an iteration.)

Gear[4] reports on a reasonably straightforward way of accomplishing this by consecutive passes over the parsed program in alternating directions. Although his technique is applicable without alteration only to explicit loops in the program with no local branching to disturb the movement of code, with more extensive flow analysis this can be generally accomplished.

Although common subexpressions other than subscript expressions can be located and moved, the benefits are not impressive and there are perils. The necessity of avoidance of altering the side effects of the evaluation is often underestimated. For this reason, more general common subexpression detection is not often done.

Since programs that do considerable accessing of multidimensional arrays spend a large part of the time in address calculation, locating common index expressions and calculating addresses recursively in loops is recommended as one of the first targets in any compiler intended to produce very high quality code.

SUMMARY

This has been intended as a brief overview of the way in which production compilers are written at the present time as seen by one of the practitioners of the art. The word art is used here as little has been accomplished in applying engineering techniques to compiler writing outside of the areas of lexical and syntactic analysis. The parts associated with declaration processing and code generation and optimization are still very ad hoc. The need for researchers to find better descriptive devices for generation strategies and the miscellaneous problems surrounding the creation of real object programs is still great. The need for compiler writers to apply the best that is already known is perhaps even greater.

REFERENCES

1 D GRIES
Compiler construction for digital computers
John Wiley 1971

2 R W FLOYD
Syntactic analysis and operator precedence
Journal of the ACM July 1963 p 316 Vol 10

3 W R McKEEMAN J J HORNING
D B WORTMAN
A compiler generator
Prentice Hall 1970

4 C W GEAR
High speed compilation of efficient object code
Communications of the ACM August 1965 p 483 Vol 8

A laboratory for the study of automating programming*

by T. E. CHEATHAM, JR. and BEN WEGBREIT

Harvard University
Cambridge, Massachusetts

INTRODUCTION

We are concerned in this paper with facilities, tools, and techniques for automating programming and thus we had best commence with discussing what we mean by *programming.* Given a precise specification of some task to be accomplished or some abstract object to be constructed, *programming* is the activity of producing an algorithm or procedure—a program—capable of performing the task or constructing a representation of the object on some computing system. The initial specifications and the resulting program are both couched in some (programming) language—perhaps the same language. The process typically involves such activities as: choosing efficient representations for data and algorithms, taking advantage of known or deduced constraints on data and algorithms to permit more efficient computations, verifying (proving) that the task will be accomplished or that the object constructed is, in fact, the one desired, demonstrating that certain performance criteria are met, and so on.

The kind of facility currently available which might be characterized as contributing to automating programming is usually called a *compiler.* It typically translates an algorithm from some higher level (programming) language to a lower level ("machine") language, attempting to utilize memory and instruction resources effectively and, perhaps, reorganizing the computational steps, as implied by the higher level language representation, to move invariant computations out of loops, check most likely (or cheapest) arms of conditions first, and so on.

We are not here concerned with traditional compilers; indeed, we will assume the existence of a good compiler. We are concerned with facilities at a "higher level": translating specifications which contain much less commitment to particular data and algorithmic representations than is usual with higher level programming languages, and performing rather more drastic reorganization of representation, implied computational steps, and even implied method of computation than is done with traditional compilers. We imagine our end product to be programs in a higher level language. On the other hand, we must note that the line between the kind of facility we will describe and a good compiler is very fine indeed and we will suggest that certain kinds of transformations sometimes made by conventional compilers might better be approached with the tools and techniques described here.

The purpose of this paper is to describe a facility which we characterize as a laboratory for the study of automating programming. We view the laboratory as a pilot model of a facility for practical production programming. The time and expense invested in programming today and the lack of confidence that most programs today actually do what they are intended to do in all cases is surely dramatic evidence of the value of such a facility. The need is particularly acute when the task to be accomplished is complex and the resulting program is necessarily large. Such situations are precisely those encountered in many research areas of computer science as well as in many production systems software projects. Dealing with this kind of complexity, which is to say producing efficient verifiably correct program systems satisfying complex requirements is a significant, decidedly non-trivial problem.

The second section of this paper contains a critical discussion of a wide variety of work and research areas which are related; the third section is devoted to a broad general description of the laboratory; the fourth section then briefly describes the ECL programming system, the intended host for the laboratory; the fifth

* This work was supported in part by the Advanced Research Projects Agency under contract F-19628-68-C-0379 and by the U.S. Air Force Electronics Systems Division under contract F19628-71-6-0173.

section discusses, in general terms, a variety of program automation techniques to be employed; the sixth section describes the basic components of the initial laboratory; and the seventh section summarizes what we hope to accomplish with the laboratory and mentions several open problems.

RELATED WORK

There is a considerable body of work and a number of current research areas which are related to programming automation. Most of this work does not, at present, provide anything like a complete system; much of it does provide *components* of a system for automating programming and is thus directly related to and sometimes directly usable in the laboratory we will describe.

We have divided the work to be discussed into seven different areas: automatic program synthesis, mechanical theorem proving, automatic program verification, program proof techniques, higher level programming languages, equivalence of program schemata, and system measurement techniques. In each case we are discussing the work of several people; the bibliography cites the recent work we feel is most relevant.

Automatic program synthesis

The basic idea here is to construct a program to produce certain specified outputs from some specified inputs, given predicates asserted true of the inputs and the outputs as related to the inputs. The basic technique is to (mechanically) prove the theorem that there exist outputs satisfying the predicate and then to extract a program from the proof for constructing the outputs. It has been suggested that these techniques can also be utilized to transform programs, for example to transform a recursive procedure into an equivalent iterative procedure using the two stage process of first deducing a predicate that characterizes the recursive procedure and then synthesizing an equivalent iterative procedure which computes the outputs satisfying the predicate deduced.

We view the work in this area to date as primarily of theoretical interest and contributing to better mechanical theorem proving and proof analysis techniques. It is often more convenient to produce an (inefficient) algorithm than it is to produce a predicate; the two stage process proposed for "improvement" of programs is awkward and, we believe, highly inefficient as compared with the direct transformation techniques to be discussed below.

Mechanical theorem proving

The heart of any system for automating programming will be a facility for mechanical theorem proving. At the present time there are two basically different approaches to mechanical theorem proving and a realization of both these approaches provide important components of our laboratory. One approach is to construct a theorem prover which will, given enough resources, prove any true theorem in the first order predicate calculus with uninterpreted constants; the other approach is to provide a theorem prover which is subject to considerable control (i.e., allows one to employ heuristics to control the strategy of proof) and which utilizes interpretations of the constants wherever possible for efficiency. Mechanical theorem provers of the first sort are now usually based on the resolution principle. We term those of the second sort "programmable theorem provers".

Resolution theorem provers

Robinson's 1965 paper introducing the resolution principle has been followed by vigorous activity in implementing mechanical theorem provers based on this principle. Much of the activity has been concerned with developing strategies for ordering consideration of resolvents; at the present time the breadth-first, unit-preference, and set-of-support general strategies have been studied and other variations are being considered. It is clear that a powerful resolution-principle based theorem prover will be an important component of the laboratory.

Programmable theorem provers

In the PLANNER system, Hewitt provides a facility for programmable theorem proving in the sense that one can very easily utilize interpretations of the objects and operations entering into a theorem to control the strategy of proof, one can make the choice of technique used in any particular instance data dependent, and can very readily employ any general mechanical theorem prover, effectively as a subroutine. The use of a small subset of Hewitt's proposed facilities by Winograd (called micro-planner by him; see [Winograd 71], [Sussman 70]) in his program for understanding natural language gives dramatic evidence of the effectiveness of the approach. We thus include a programmable theorem prover on the style of Hewitt's and Winograd's as the basis for the theorem proving component of our laboratory.

Automatic program verification

The work in this area is concerned with mechanization of what we term "flow chart induction". Given a representation of some algorithm as a flow chart with assignments, conditional branching, and looping, one appends to the boxes of the flow chart predicates asserted true of the variables at various points in the computation and, in particular, of the inputs and the outputs. The procedure is then to attempt to mechanically demonstrate that the whole is consistent and thus that the program is correct.

Again, we view this work as primarily of theoretical interest. The theorem proving techniques utilized in King's system (see [King]) are particularly interesting, however, as they utilize interpretations of the integers and operations over the integers; while not general, they do provide rather more efficient methods for proofs concerning integers than is presently possible with the more general resolution-type proof methods which do not employ interpretations.

Program proof techniques

A number of workers have been concerned with developing proof techniques which are adapted to obtaining proofs of various properties of programs. These include some new induction methods—structural induction and flow chart induction—simulation techniques, and so on. This work provides a very important basis for proving the equivalence of various programs.

Higher Level Programming Languages

A considerable amount of work in the development of higher level programming languages has been concerned with providing languages which are particularly appropriate for certain application areas in the sense that they free the programmer from having to concern himself with the kind of detail which is not relevant to the application area in which he works. For example, APL permits a programmer to deal with arrays and array operations and relieves him of concern with the details of allocation, accessing, and indexing of the array elements. SNOBOL 4 directly supports algorithms which require back tracking, providing the mechanics automatically and permitting one to write theorem-proving, non-deterministic parsing, and such like algorithms very easily. SETL provides general finite sets as basic data objects plus the usual array of mathematical operations and predicates on sets; it thus permits one to utilize quite succinct statements of a wide variety of mathematical algorithms and to considerably ease the problem of proving that the algorithms have certain properties. ECL (which we discuss in some detail in a later section) provides a complete programming system with facilities which permit one to construct extended language facilities such as those provided in APL, SNOBOL 4, and SETL, and to carefully provide for efficient data representation and machine algorithms to host these extended language facilities.

Equivalence of program schemata

There has been considerable interest recently in studying various program schemata and investigating their relative power, developing techniques for proving their equivalence, and so on. Most of the work to date is interpretation independent and while, for example, many transformations from recursive specification to iterative specification of algorithms have been developed, it is clear that many practical transformations cannot be done without employing interpretations.

The most common use of interpretation dependent transformations is in "highly optimizing" compilers. There, a very specific, usually *ad hoc* set of transformations is employed to gain efficiency. Often the transformations are *too ad hoc*—under certain conditions they do not preserve functionality (i.e., don't work correctly).

System performance measurement techniques

It is a quite straightforward matter to arrange for various probes, monitors, and the like to permit measurements of the performance of programs, presuming that appropriate test input data sets are available, and a considerable amount of work has been done in this area. However, there are two further areas which are now the subject of investigation which we feel may yield important components for our laboratory. These are mathematical analysis of algorithms and automatic generation of probability distributions for interesting system parameters.

Mathematical analysis of algorithms

Several people have been working recently in the area of developing methodology and techniques for the mathematical analysis of algorithms to obtain estimates and/or bounds on certain critical parameters and for developing (and proving) optimal algorithms for certain functions. We envision the manipulation facilities of the laboratory as being readily adaptable to providing mechanical assistance in this activity, particularly in the area of aiding in the inevitable symbolic algebraic manipulation required in carrying out a mathematical analysis.

Automatic synthesis of probability distributions

Some recent work by Nemeth (see [Nemeth]) may, when it is developed further, provide an interesting and valuable component of the laboratory. What he is trying to do is to develop algorithms for mechanically generating probability distributions for various parameters from a computational schema augmented by given probability distributions for input variables and functions employed. Use of techniques like his should prove far superior to actually carrying out the computation for sample data values. A mixture of mechanical generation of distributions and carrying out portions of a computation might, in the earlier stages, provide a practical tool.

All the above work is related and relevant to automating programming, but none, in our opinion, is adequate alone. The need now is to integrate these facilities, techniques, and so on into a *system*—a laboratory for the study of automating programming.

THE APPROACH TO BE TAKEN IN THE LABORATORY

The goal of such a laboratory is a practical, running system that will be a significant aid to the construction of real-world programs. Automating programming entails transferring to the computer those facets of programming which are not carried out efficiently by humans. It is our contention that the activity most in need of such transfer is the optimization (in a very broad sense of the word) of programs. The orientation of the laboratory and the principal task to which it will be put is that of taking an existing program and improving upon it.

That optimization is, indeed, a key problem requires little defense. If "program" is taken in a sufficiently broad sense, it is easy to produce *some* algorithm which performs any stated task. Given just the right language, program synthesis is seldom a significant issue. For many problems, the most natural task description is precisely a program in an appropriate notation. The use of an extensible language makes straightforward the definition of such notation. For other problems, it may be that a predicate to be satisfied is a better task statement, but this too is in some sense a program. The line between procedural and non-procedural languages is fuzzy at best and it may be erased entirely by the use of theorem-proving techniques to transform predicates into programs (and conversely).

As we see the problem, the issue is not arriving at a program, but arriving at a good one. In most cases, programs obtained from theorem provers applied to predicates, from a rough-cut program written as a task description, or even from the hands of a good programmer leave much to be desired. Often, the initial program is several orders of magnitude away from desired or even acceptable behavior. The larger the program, the more likely this is to be the case. The reasons are generally such defects as inefficient representation of data, failure to exploit possible constraints, use of inefficient or inappropriate control structures, redundant or partially redundant computations, inefficient search strategies, and failure to exploit features of the intended host environment. Recognizing the occurrence of such defects and remedying them is the primary goal of the laboratory.

ECL AS A BASIS FOR THE LABORATORY

The ECL programming system and the EL1 language have been designed to allow rapid construction of large complex programs, perhaps followed by modification and contraction of the programs to gain efficiency. The facilities of ECL permit one to compose, execute, compile and debug programs interactively. The EL1 language is an extensible language with facilities for extension on three axes: syntax, data, and operations.

The EL1 language plays four roles in the laboratory: (1) it is the language used to construct the various components of the system; (2) it and its extensions are the language used to state algorithms which are to be manipulated by the system; (3) it is the target language for transformations (i.e., EL1 programs are transformed into better EL1 programs); and (4) it is the host language for the theorems constituting the data base.*

The features of EL1 and its host system, ECL, which are particularly relevant to the laboratory are the following:

(a) Data types (called "modes" in EL1) can be programmer defined using several basic data types (e.g., integers, reals, characters, etc.) and, recursively, several mode valued functions (e.g., construction of homogeneous sequences, nonhomogeneous sequences, pointers, etc.).

* That is, the parts of a theorem (i.e., the conditions, antecedent, consequent, and recommendation list as described in the following section) are couched in an extension of EL1 and "glued together" as an EL1 procedure by operators defined as extensions; of course, the theorems are not "executed" in the usual sense.

(b) Procedures can be *generic* in the sense that a given procedure (say that for +) can have a number of different bodies or meanings and the selection of a particular body or meaning to be used is determined by the mode(s) of the argument(s) utilized in some call of the procedure. Thus, for example, it is particularly straightforward to accommodate refinements in representation of some general data type (e.g., sets) without doing violence to algorithms defined on it by associating a new mode with a refinement and defining new versions of operators particularized to that mode via the generic mechanism.

(c) The compile-time, load-time, run-time, and the like kinds of restrictions employed in most systems are not present in ECL. In particular, the ECL compiler is a program which can be called at any time; it is given a program to be compiled and a list of variables (free in that program) whose values are to be taken as fixed. It then converts the program to a form which takes advantage of whatever efficiencies it can from the freezing of values. There is no distinction between compiled code and non-compiled (interpretive) code insofar as their discernible effect when executed.

(d) ECL provides storage allocation and reclamation mechanisms which are quite sensitive to space/time efficiencies. Thus, the so-called "data compiler" goes to some lengths to utilize memory efficiently when allocating a component of a complex data structure containing several "pieces".
The use of both "stack" and "heap" mechanisms for allocation and freeing of space is also provided.

(e) ECL provides for multiple concurrent paths and permits complete user control over the environment for each concurrent path. In addition to permitting strategies such as employing, say, a general resolution theorem prover to be applied to some difficult theorem in parallel with other undertakings, this feature makes it particularly straightforward to set up and run some program in an environment of ones choosing; for example, to gather statistics on its behavior in some simulated "real" environment.

(f) Error conditions (both system detected and those defined by user programs) are generally handled by setting interrupts and each interrupt is, optionally, handled by a user defined procedure. This feature is very helpful in test execution and evaluation of subject programs and permits construction of elaborate control mechanisms when appropriate, as might be the case in controlling the behavior of the programmable theorem prover.

There are two extensions of ECL which provide particularly convenient linguistic facilities for stating algorithms; these are an extension which permits one to deal with sets and set operations* and an extension which hosts non-deterministic programs.

OPERATION OF THE INITIAL LABORATORY

The laboratory, like ECL, is intended for interactive use. A programmer approaches it with a problem specification and a set of requirements on how the program is to perform. He and the system together undertake to produce a program which meets the specifications and requirements. The intention is that the laboratory be a practical tool for everyday use, i.e., that hard, real-world problems with realistic performance requirements be brought to and handled by the laboratory.

The problem specification may be an existing program written in EL1, possibly a long-standing production program. In this case, the presumption is that its performance falls short of that required and that the concern is with automating its tuning. Alternatively, the specification may be an EL1 program written in a very liberal extension and constructed solely as a concise algorithmic statement of the problem task. There may be little expectation that such a program will meet any non-trivial performance requirements. Significant improvements may be needed even to reach the desired order of magnitude. Finally, the problem specification may be a set of predicates to be satisfied. Here the laboratory begins by constructing an initial EL1 program using as its target an extension set designed for this purpose. Again, such a program may be several orders of magnitude removed from acceptable performance.

In very general terms, the laboratory is a man-machine system for transforming the initial program to an equivalent one which meets the stated requirements. The heart of the system is a set of transformations, actually theorems concerning the language, which preserve functionality while improving the program. Deciding whether an arbitrary transformation either preserves functionality or improves the program is, of course, impossible, but decision procedures for the gen-

* This extension permits us to capture most of the facilities proposed for the language SETL. See [Schwartz 70] for a particularly cogent argument for providing sets and set operations in a higher-level programming language.

eral case are not needed here. The laboratory will employ specific transformations which under appropriate circumstances—i.e., when their enabling predicates hold—maintain program equivalence. Constructing these transformations and verifying that the validity of the enabling predicates do insure functionality will be a task assigned to humans who may, of course utilize the facilities of the laboratory to prove the validity. While the *functionality* of the transformations may be assured, such is not the case for their *effectiveness.* To obtain a sufficiently powerful set of transformations it is necessary to include many whose utility is conditional, e.g., those which are effective only under circumstances which are difficult or impossible to verify analytically, or those which optimize performance in one metric at the (perhaps unacceptable) expense of another. In general, the transformation set will include transformations which are mutually exclusive (i.e., only one of some subset can be applied) and some which are inverses (i.e., applying two or more repeatedly will lead to a loop). Hence, choice of which transformations to apply is governed specifically by the performance requirements demanded of the program and the disparity between these and the program at each state of optimization.

Determining program performance is a crucial issue. There are two basic approaches, both of which are used in the laboratory. The first is *analytic.* The system derives closed-form expressions for program behavior based entirely on a static inspection of program structure and interpretation of the program operations. Then given a description of an input data set, e.g., as a set of probability distributions for possible input values, the system can describe the exact program behavior. Whenever such closed form expressions can be obtained, this is clearly the best certification of program performance. However, at present our analytical techniques are too weak for any but the simplest programs. The second approach is that of actually running the program on benchmark data sets, data sets provided by the programmer as part of his performance specifications. Between these two extremes lies the spectrum of simulation: those portions of the program which can be treated analytically are replaced by simulation blocks and the rest of the program is run as is. The large area of mixed strategy is particularly powerful since it allows one to use only partially representative benchmark data sets yet extrapolate meaningful results from them by the use of analytical techniques.

The utility of the laboratory will be governed principally by the specificity of admissible performance specifications and the degree to which they can be met on the intended machine. Performance specifications include the obvious bounds on execution time and space. Alternatively, they might be cast in the form: as fast or as small as possible. This is, however, only a rough cut. Few problems and fewer host machines are entirely homogeneous. In real time situations, optimizing total execution time may be far less important then attaining a certain minimum for particular sections. Similarly, the total space occupied by a program and its data is far less important than the distribution of this space over several storage devices of various capacities and access speeds. Also, the intended host machine may be a multiprocessor or provide multiprocessing capabilities by means of special processors (e.g., graphics) or remote processors (e.g., a network). Partitioning the computation among the various processors so that the computation load on each is beneath prescribed limits is another task of the laboratory.

The possible transformations for obtaining the desired performance vary considerably in scope, power, and effectiveness. A sketch of those which currently seem to have the greatest payoff may give the flavor of what may be accomplished.

The most straightforward are those for reducing the space occupied by data. Any field containing literal data restricted to N possible values requires, of course, no more than $[\log_2 N]$ bits. What may not be quite so obvious is that with an appropriate programming language, simply changing declarations is all that is required to control the storage allocated, perform the coding and uncoding on data access, and handle the necessary conversions. EL1 is such a language; hence, the class of transformations is readily obtained. In the case of sequences of literal data fields (e.g., character strings), further compression can be obtained by block encoding. Relations can be represented in a variety of ways and changing from one to another often results in significant economics. Sparcely populated arrays can be changed to lists or hash tables in which the array indices are retrieval keys. Conversely, the space occupied by pointers (e.g., in list structure) can be reduced if linked lists are replaced by arrays in which relations are represented by small integer array indices occupying only a few bits.

One candidate for optimization of both time and space is sets and set operations. There are a number of particularly efficient representations for sets applicable only under certain conditions (e.g., bit vectors when the number of elements is fixed and relatively small or lists in canonical set order when the generation of new sets is carefully controlled) or efficient only when the set operations are circumscribed (e.g., hash tables when the operations are union and intersection but not set complement). When such a representation is possible, its use will often produce dramatic improvement over standard list structure techniques.

Transformations for optimizing time are often subtle and require sophisticated techniques for manipulating program structures. Perhaps the best understood sort is the transformation of recursive to iterative programs. Even restricting attention to uninterpreted schemas, there are several interesting schema classes for which the transformation can always be carried out. By adjoining additional transformations which exploit the properties of specific common program operations, a very powerful tool for eliminating program recursion may be obtained.

Time can always be saved at the expense of space by substituting the definition of a routine for a call on it. Where recursion is absent or has been previously removed, it is possible to perform repeated back substitution until all calls have been eliminated. While too costly in space to be employed everywhere, it is very effective if performed on the most frequently executed portions of the program. Aside from the obvious virtue of eliminating the expense of function call, it has the more significant virtue of allowing each back substituted defining instance to be optimized independently—in the context of the text into which it is placed.

The principal class of time optimizations is the elimination of searches. A few such transformations are simple, e.g., the replacement of association lists by hashed structures or balanced trees, and the substitution of arrays for lists which are frequently accessed by indexing. Searching is, however, not confined to lists. Structures of all sorts—arrays, queues, strings, and so on—are frequently searched for an element or set of elements having some property. When the density of hits is small, blind search is inefficient. An appropriate method is to adjoin bookkeeping records to the given structure and add bookkeeping code to the program so as to keep track of what would be the result of searching. When the incremental costs of maintaining the necessary records is small compared to the search cost, this often provides a significant optimization. Determining what records must be kept and how to keep them are non-trivial problems, but ones which appear open to solution at least for many significant program classes.

A related class of program optimizations is based on reordering computations in operations on compound structures. Any operation on a compound object must be defined in terms of primitive operations or its primitive components. Given a sequence of operations on compound objects, it is usually possible to reorder the underlying sequence of primitive operations to reduce some computational resources. Galler and Perlis (see [Galler 1970]) discuss in detail the problem of saving the storage required for temporaries in matrix operations and mention that a variation on their technique can be used to minimize time. It appears possible to generalize these results to arbitrary data structures. First, recursion is eliminated from function definitions. Then each compound operator is replaced by its definition until only primitive operators appear (i.e., the back substitution optimization mentioned above). Then, program loops are merged to carry as many operations as possible on each loop. Finally, dependency analysis is used to find common sub-expressions and eliminated unnecessary temporaries. Any number of *ad hoc*, type dependent, transformations can be added to this basic framework. The basic technique, that of unwinding compound operations and then winding them up again in a more optional fashion, is broadly applicable.

Several sets of transformations are concerned with effective utilization of the particular host machine(s). These are therefore specific to the environment, but no less important than their general cousins. The most important is that of partitioning the computation among several processors. In so doing, the first step is to take a conventional sequential program and transform it to a representation which exhibits all the potential parallelism so that sequencing is dictated only by data dependency. The next step is to partition the transformed program among the available processors in such fashion that (1) upper bounds on computational resources demanded of each machine are obeyed; (2) communication between processors is carried out satisfactorily along the available data paths and (3) the entire configuration has the desired performance characteristics. Item (2) can be reduced to item (1) by treating each data path as a processor with its own (perhaps small) computational bounds. Item (1) is, of course, the heart of the matter. The work of Holt, Saint, and Shapiro provides a very promising approach to this and has already demonstrated some success in certain restricted applications (see [Shapiro 69]).

This set of program transformations is only representative. Others will be added in time as users of the laboratory gain practical experience with and understanding of program transformation. However large such a collection, it is only a beginning and its exact composition is only a secondary issue. More important is the problem of determining which transformations to use and where, given a program and a set of performance specifications. The first step is to refine the performance measurements so as to determine precisely where the specifications are not being met. Correlating the various measurements with program text is straightforward.

Given the program text augmented with resource utilization statistics, the next task of the laboratory is to find places in need of optimization, find one or more appropriate transformations, verify whether they are applicable and apply them. In choosing places to concentrate attention the laboratory starts simply by looking for the areas with the largest cost relative to that desired. Given these obvious starting places, the key problem is tracing back from these when necessary to the causes, perhaps at some point far removed in the program. In this step, and in the choice of transformation classes to be attempted, there will be the opportunity for explicit guidance by the programmer. If no guidance is given, the laboratory will doggedly pursue possible hypotheses but the search may be cut dramatically by human intervention.

Even with this assistance, the proposed transformations must be taken as tentative hypotheses to be explored. Few transformations always result in improvement. Many optimize one resource at the expense of others, while some transformations are of use only for certain regions of the data space. Hence, in general, it is necessary to continually verify that the transformed versions of the program are indeed improvements. Again, program analysis coupled with performance measurement tools will be employed.

In summary, the principal functions of the laboratory are choosing areas of the program to be optimized, carrying out pattern matching to determine the applicability of various transformations, performing these transformations, and arranging for explicit guidance by the programmer in this process. The laboratory consists of a set of components for carrying out these activities in concert.

COMPONENTS OF THE INITIAL LABORATORY

As we noted previously the laboratory is, essentially, one particular extension of ECL and that the internal representation for programs and data employed in ECL will be utilized for programs being manipulated (synthesized, transformed, proved to have some property, and so on). Here we will describe the several components in the initial laboratory—the several EL1 programs and/or ECL extensions which together constitute the initial laboratory. Before launching into the discussion, however, we want to note the influence of the work of Hewitt and Winograd on ours; the basic components of the laboratory have very much the flavor of the linguistic-independent components of Winograd's version of Hewitt's PLANNER system. Our discussion of the components is quite brief as our intention in this paper is to provide an overview of the laboratory and the general approach being taken. A detailed discussion of the current versions of the components is provided in a paper by Spitzen (see [Spitzen 71]).

Control

There is a top-level control path* which provides the interface between the user and the laboratory. It provides for input of the program to be manipulated and the associated performance criteria, if any. It then arranges that the program to be manipulated is put into a canonical form and sets up a parallel path which provides the manipulation strategy by calling for the appropriate theorem or theorems to be applied. The control path then provides for dialogue between the system and the user so that the system can request information from the user (e.g., verification that certain constraints hold in general, test data, and the like).

Programmable theorem prover

A "theorem" in the sense this term is used in the system consists of a condition which governs the applicability of the theorem, an antecedent, a consequent, and a recommendation list. Given some program structure, an instance of substructure matching the antecedent can be replaced by an equivalent substructure corresponding to the consequence so long as the condition holds. The recommendation list is basically a list of predicates which govern the use of other theorems to be used in manipulating the structure to match the antecedent in applying this one and it is the careful use of this "governor" which permits the theorem proving to operate in a practicable time frame. Note that in a very strong sense theorems are really programs effecting transformations which, through the conditions and recommendation lists (arbitrary EL1 predicates), can establish elaborate control and communication arrangements.

Data base

The theorems which constitute the basis for the transformations and manipulations performed by the system are housed in a data base. There is a continually growing static component of the data base which includes the "standard" theorems. In addition there may be a collection of theorems appropriate to certain particular investigations with which we will augment the data base at appropriate times. There is also a dynamic component which varies as a program is being manipulated. For example, if we commence dealing with the arm "$p_2 \Rightarrow e_2$"

* ECL provides multiprogramming; "path" is the ECL terminology for a parallel path of control.

of the conditional expression

$$[p_1 \Rightarrow e_1;\ p_2 \Rightarrow e_2;\ \ldots;\ p_n \Rightarrow e_n]$$

Then we would add the theorem appropriate to "$\neg p_1$" to the data base and arrange that it be removed when we get to the stage of treating the conditional expression as a whole in "higher level" manipulation of the program structure containing it.

The data base is quite large and, as various new facilities are added to the system, the data base will grow. Its size plus the fact that one wants to be able to select appropriate theorems (i.e., in accordance with a given recommendation list) from the data base efficiently, make it imperative that a certain amount of structure be imposed on the data base. Initially this structure basically amounts to a collection of "threads" through the data base, implemented by a combination of list structure, hash, and descriptor coding techniques. It is anticipated that getting an efficient structuring of the data base as it grows will pose a non-trivial research problem which we anticipate being attacked with the tools provided by the laboratory itself.

Pattern Matcher

The process of applying some theorem to some program structure involves matching the parts of the antecedent to acceptable corresponding parts of the structure; in general, of course, this will involve calls on other theorems to manipulate the structure into an appropriate format and/or the verification that certain conditions maintain. It is the pattern matcher which administers this activity; it will make tentative part-part matches, reject matches or attempt them in new ways when subsequent failure to match occurs, and so on.

Backtracking mechanism

The pattern matcher operates as a non-deterministic program in the sense that it makes provisional matches which must be "unmatched" and rematched when failure occurs to match some subsequent part of the antecedent to the program structure being manipulated. A backtracking mechanism must therefore be provided so that the effects of computations and variable bindings can be "undone." The method we use to accomplish this is to alter the basic EL1 evaluator so that, when backtrackable code segments are executed, any change to the environment is preceded by recording the appropriate modification to the environment which will undo the change in the event backtracking is required.

Measurement techniques

In addition to the usual facilities for inserting probes to provide measures of utilization of various functions and data elements provided in ECL and the usual ability to obtain measurements by sampling driven by a real-time clock there are two measurement components in the system which are less usual. Precise delineation of potential inefficiencies sometimes requires very exact timing data. Unfortunately it is usually impossible to get very fine-grained timing from most contemporary machines. Hence, the laboratory includes a machine language simulator for its host machine, i.e., a program which executes machine code interpretively and gathers statistics as requested. This permits programs to be run unchanged while collecting very precise data on the distribution of program execution time and storage requirements. This data, combined with that obtained by inserting measurement probes into the program permits performance measurements to be made to any level of detail. The second measurement component is an implementation of the "probability distribution computer" described by Nemeth.

GOALS OF THE LABORATORY

It must be stressed that the laboratory is intended for practical use, for the attainment of useful results which would be difficult to obtain without its assistance. It is the deliberate intention that one be able to approach it with a non-trivial program and obtain with it a significantly improved version. Such programs will include applications programs, system programs such as the EL1 compiler, as well as modules of the laboratory itself.

It appears that this will be achievable even with the simple approaches to be taken in the initial version of the laboratory. The use of *programmable* theorem provers and search techniques have made it possible to quickly endow the laboratory with considerable expertise, if not deep insight. On large production programs (which commonly are so complex that no one really understands their exact behavior) the laboratory may be expected to make significant improvements by more or less mechanical means. Such mechanical skill is precisely what is required to complement the high level programmer. Initially, the laboratory will serve as a programmer's assistant, suggesting possible areas of improvement and carrying out the transformations he chooses, but leaving the creative work to him. Even at this level the laboratory may serve better than the average junior programmer. Further, the initial laboratory will serve as a basis for developing more sophisticated control over the choice of transformations to be applied.

A topic which falls out as another application of this work is the modification or adaptation of programs. Given a program which does a certain job well, it is very common that it must be modified to handle a related task or work under changed conditions. Usually such modifications are done by hand, seldom with entirely satisfactory results. Hence, automation of the process is attractive. One method which has been occasionally proposed for performing this is to construct a system which when fed the existing program, deduces how it works, and then performs the specified modifications. This requires the system to dig the meaning and purpose out of the existing program—an operation which is difficult at best and perhaps impractically expensive. A solution which shows greater promise is to use the laboratory to combine man and machine. If programs are developed using the laboratory, then for any production program, there is an initial program from which it was derived. Given a statement of the new task to be accomplished, the programmer can make his modifications to the relatively transparent and simple initial program. This is still hand labor, but at a much higher level. Applying the laboratory to the optimization of the modified program results in a production program with the desired properties. This puts creative work in adapting a program into the hands of the programmer while freeing him from the drudge work.

It is anticipated that as the programmer gains experience with the system, he will develop his own set of optimization techniques. By making it convenient for him to add to the data base these transformations along with patterns for recognizing potential situations for their applications, we allow a certain degree of growth in the expertise of the laboratory. Given a clever population of users, the laboratory should grow to some considerable level of sophistication, at least in areas of interest to that user population. This scenario has, of course, its limitations. Merely expanding the data base would, in time, cause the system to flounder in a sea of possible but irrelevant transformations. Hence growth of the system must ultimately depend on significant improvements in global strategies, local heuristics, and theorem provers. In particular, the need for specialized theorem provers will very likely arise.

At present, it is not clear how to proceed in these directions, nor is this surprising. One of the purposes of the laboratory is to gain expertise in program manipulation, determine the limitation of current techniques, and advance to the point where the real problems can be seen clearly. The initial laboratory is a first step, but a significant step, in this direction.

BIBLIOGRAPHY

E A ASHCROFT (1970)
Mathematical logic applied to the semantics of computer programs
PhD Thesis Imperial College London

E A ASHCROFT Z MANNA (1970)
Formalization of properties of parallel programs
Stanford Artificial Intelligence Project Memo AI-110 Stanford University

R M BURSTALL (1970)
Formal description of program structure and semantics in first-order logic
Machine Intelligence 5 (Eds Meltzer and Michie)
Edinburgh University Press, 79-98

R M BURSTALL (1969)
Proving properties of programs by structural induction
Comp J 12 1 41-48

D C COOPER (1966)
The equivalence of certain computations
Computer Journal Vol 9 pp 45-52

R W FLOYD (1967)
Assigning meanings to programs
Proceedings of Symposia in Applied Mathematics American Mathematical Society Vol 19 19-32

R W FLOYD (1967)
Non-deterministic algorithms
JACM Vol 14 No 4 (Oct.)

B A GALLER A J PERLIS (1970)
A view of programming languages
Chapter 4 Addison-Wesley

C GREEN (1969b)
The application of theorem proving to question-answering systems
Ph D Thesis Stanford University Stanford California

C GREEN B RAPHAEL (1968)
The use of theorem-proving techniques in question-answering systems
Proc 23rd Nat Conf ACM Thompson Book Company Washington DC

P J HAYES (1969)
A machine-oriented formulation of the extended functional calculus
Stanford Artificial Intelligence Project Memo 62 Also appeared as Metamathematics Unit Report University of Edinburgh Scotland

C HEWITT (1971)
Description and theoretical analysis of plannar
Ph D Thesis MIT January

C B JONES P LUCAS
Proving correctness of implementation techniques
Symposium on Semantics of Algorithmic Languages Lecture Notes in Mathematics 188 New York

D M KAPLAN (1970)
Proving things about programs
Proc Princeton Conference on Information Science and System

D M KAPLAN (1967)
Correctness of a compiler for algol-like programs
Stanford Artificial Intelligence Memo No 48 Department of Computer Science Stanford University

J KING (1969)
A program verifier
Ph D Thesis Carnegie-Mellon University Pittsburgh Pa

J KING R W FLOYD (1970)

Interpretation oriented theorem prover over integers
Second Annual ACM Symposium on Theory of Computing
Northampton Mass (May) pp 169-179

D C LUCKHAM N J NILSSON (1970)
Extracting information from resolution proof trees
Stanford Research Institute Artificial Intelligence Group
Technical Note 32

Z MANNA (1969)
The correctness of programs
J of Computer and Systems Sciences Vol 3 No 2 119-127

Z MANNA (1969)
The correctness of non-deterministic programs
Stanford Artificial Intelligence Project Memo AI-95 Stanford
University

Z MANNA J McCARTHY (1970)
Properties of programs and partial function logic
Machine Intelligence 5 (Eds Meltzer and Michie)

J McCARTHY (1963)
A basis for a mathematical theory of computation
Computer programming and formal systems
(Eds Braffort and Hirschberg) Amsterdam North Holland
pp 33-70

J McCARTHY J A PAINTER (1967)
Correctness of a compiler for arithmetic expressions
Mathematical Aspects of Computing Science Amer Math Soc
Providence Rhode Island pp 33-41

R MILNER
An algebraic definition of simulation between programs
Stanford Artificial Intelligence Memo AI-142

A NEMETH
Unpublished; to be included in his Ph D Thesis Harvard
University

J A PAINTER (1967)
Semantic correctness of a compiler for an algol-like language
Stanford Artificial Intelligence Memo No 44 (March)
Department of Computer Science Stanford University

M S PATERSON (1967)
Equivalence problems in a model of computation
PhD Thesis Cambridge University

G ROBINSON L WOS
Paramodulation and theorem-proving in first-order theories with equality
Machine Intelligence Vol IV Ed by D Michie

J ROBINSON (1966)
A review of automatic theorem proving
Annual Symposia in Applied Mathematics Vol XIX

J T SCHWARTZ (1970-71)
Abstract algorithms and a set-theoretic language for their expression
Preliminary Draft First Part Courant Institute of
Mathematical Sciences NYU

J M SPITZEN (1971)
A tool for experiments in program optimization
(in preparation)

R M SHAPIRO H SAINT (1969)
The representation of algorithms
Rome Air Development Center Technical Report 69-313
Volume II September

H A SIMON (1963)
Experiments with a heuristic compiler
JACM (October 1963) 482-506

J SLAGLE (1967)
Automatic theorem proving with renameable and semantic resolution
JACM Vol 14 No 4 (October 1967) 687-697

J SLAGLE (1965)
Experiments with a deductive question-answering program
Comm ACM Vol 8 No 12 (December) 792-798

H R STRONG JR (1970)
Translating recursion equations into flow charts
Proceedings Second ACM Symposium on Theory of Computing
(May) pp 184-197

G J SUSSMAN T WINOGRAD (1970)
Micro-plannar reference manual
MIT AI Memo No 203 (July)

R J WALDINGER (1969)
Constructing programs automatically using theorem proving
PhD Thesis Carnegie-Mellon University (May 1969)

B WEGBREIT (1971)
The ECL programming system
Proc FJCC Vol 39

R WINOGRAD (1971)
Procedures as representative for data in a computer program for understanding natural language
Project MAC MIT MAC-TR-:4 February

Segmentation and optimization of programs from cyclic structure analysis

by JEAN-LOUP BAER and ROBERT CAUGHEY*

University of Washington
Seattle, Washington

INTRODUCTION

Modelling of computer programs by directed graphs, where the vertices represent the computational tasks and the arcs show the flow of control, has been used for optimization purposes,[1,10] parallel processing evaluation,[2,8] and segmentation.[7,9,13] All these studies are mainly based on the fact that a high proportion of the execution time of a program is spent in loops. Although the cyclic structure of programs can be theoretically complex, it has been observed that the nestedness is seldom very deep.[6] Thus if one wants to optimize programs written in a high-level language, the detection of cycles by the compiler and its use in an optimization phase may yield a definite improvement in execution time without having to pay too heavily for excess compiling time. In this paper we show how a compiler could make use of an efficient cycle detection algorithm[12] to model the embeddedness of cycles by an acyclic directed graph. The latter can then be used for packing purposes in case of a paging or "cache" memory system as well as for predicting the expected number of executions of a given statement for optimization purposes.

MODELLING OF FORTRAN PROGRAMS

For clarity of exposition we present the modelling process as a sequence of four phases. However, actions of phases 1, 2, and some of 3 are performed concurrently. The source program itself is scanned only once. The four phases are now described both functionally and in terms of the basic data structures. Detailed algorithms can be found in Reference 4.

Phase 1

Directed graph model—Instruction Sequences

The basic model is a *directed graph* $G(W, U)$ where W is the set of vertices $\{w_1, w_2, \dots, w_m\}$ and U is the set of ordered pairs (or arcs) $u_k = (w_i, w_j)$. Vertices represent computational tasks and arcs show the flow of control. Methods to analyze a FORTRAN source program and transform it into an equivalent—in terms of control—directed graph are now well-known. We shall follow Russell's approach[10] and therefore we assume the uniqueness of an initial vertex w_1 and terminal vertex w: as well as connectedness of the graph. Appendix I shows the vertices and arcs generated from the analysis of the current source statement according to its type. Figure 1 gives an example program (executable statements only) and its directed graph representation. Associated with each vertex w_i is a volume attribute v_i which represents the amount of memory words required for the machine code generated by the translated statement.

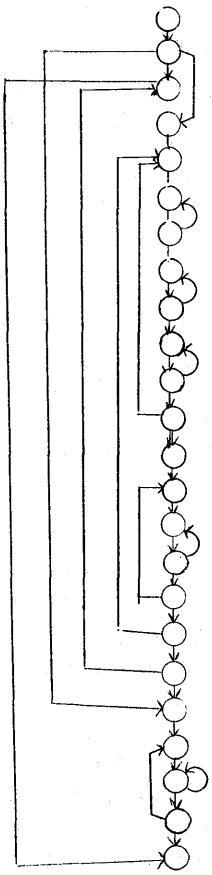
Thus given a source program, the first part of the modelling process yields $G(W, U)$ which can also be conveniently represented by an $m \times m$ (where $m = |W|$) Boolean connectivity matrix C,[9] and a volume vector $V(|V| = m)$.

For reasons which will become apparent in Phase 3 we now define the *Instruction Sequences* (I.S.) of a source program.

Let A be the class of all FORTRAN executable source statement types which do not have the attribute of implicit flow of control* and $\bar{A}$ its complement with respect to the class of FORTRAN executable source statements. Let $Z = S_1, S_2, \dots, S_i, \dots, S_n$ be the

* Present address: System Services Department, Snohomish County, Everett, Washington.

* I.e., these statements which have as a unique immediate successor the next statement in sequence.



```
 1          READ (5,5) K1,I1,I2,I3
 2          IF (K1) 10,20,300
 3    10    STOP
 4    20    K1=1
 5    23    K1 = K1 + 1
 6    25    I1 = I1 + 1
 7          IF (I1) 25,30,30
 8    30    I2 = I2 + 1
 9          IF (I2) 30,35,35
10    35    I3 = I3 + 1
11          IF (I3) 35,40,40
12    40    IF ((I1+I2+I3).LT. 15) GO TO 23
13          DO 4000 I = 1,50
14  3900    M(I) = (I*I1 + K1)
15  3995    M(I) = 2*M(I)
16          IF (M(I).LT. 25) GO TO 3995
17  4000    CONTINUE
18          IF (K1 - 15) 23,400,400
19   400    IF (K1 - 50) 300,300,10
20   300    DO 200 J = 1,10
21          DO 195 I = K1,50
22   195    N(I,J) = 2*I/3*J
23   200    CONTINUE
24          END
```

Figure 1—Model of a FORTRAN program (executable statements only)

finite string of source statements which comprise a program, in the order (on ascending index) in which they are initially submitted to a translator. An instruction sequence I_k is a non-empty substring of consecutive elements of Z written as

$$I_k = S_{k1}, \ldots, S_{km}$$

such that:

(i) $S_{kj} \in \bar{A}$ for $1 \leq j \leq m-1$ and S_{k1} is called the initial statement of I_k

(ii) $S_{km} \in A$ and S_{km} is called the terminal statement of I_k

(iii) I_k is the substring of Z of maximum length which contains S_{km}.

Z is partitioned into instruction sequences and can now be rewritten as:

$$Z' = I_1, I_2, \ldots, I_i, \ldots, I_p.$$

Z' can also be represented by a directed graph $H(I, Y)$ where I is the set of vertices I_i representing the I.S.'s and $y_k = (I_i, I_j)$ is an arc $\in Y$ if the terminal statement S_{im} of I_i can transfer explicitly to a statement S_{jp} of I_j. This directed graph can also be represented by a Boolean connectivity matrix M.

Given Z the string of source statements, the determination of the I.S.'s and their interconnections is easy and not time consuming (Figure 2 displays these new structures for the example program of Figure 1). It follows the same algorithms as the one used for the generation of branch instructions by the compiler.

Phase 2

Identify DO-loops and their embeddedness

In this phase we record the membership of statements in DO-loops in the following way. Each DO-loop is represented by an m-dimensional Boolean vector, say d_i for the ith DO-loop, such that $d_{ik} = 1$ if the statement represented by vertex w_k is in the DO-loop and $d_{ik} = 0$

I_1 = (1,2)

I_2 = (3)

I_3 = (4,5,6,7)

I_4 = (8,9)

I_5 = (10,11)

I_6 = (12)

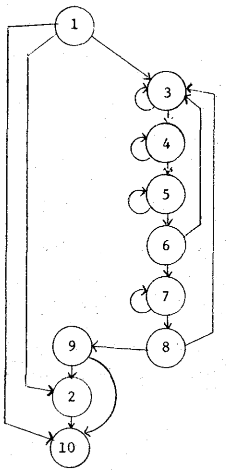
I_7 = (13,14,15,16)

I_8 = (17,18)

I_9 = (19)

I_{10} = (20,21,22,23,24)

(a) Instruction Sequences.



(b) Graph H(I,Y)

$$\begin{array}{c|cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
3 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
8 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
9 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

(c) Matrix M

Figure 2—Alternate representations of the example FORTRAN program

otherwise. At the same time, we build a (partial) embeddedness Boolean matrix E^+ which is such that $e_{ij}^+ = 1$ if DO-loop j is nested within DO-loop i.

The construction of the d_i's and E^+ is up to this point straightforward and mainly requires the same techniques as those used generally in the compilation of DO-loops, that is the maintenance of a stack of currently active DO-loops and their associated terminating labels. Figure 3 shows the d_i's and E^+ after DO-loop processing for the example of Figure 1.

Phase 3

Identify the remaining elementary cycles

A general algorithm to detect the elementary cycles in a directed graph has been recently published.[12] It is based on a path extension technique with each vertex of the graph being a potential starting point for a cycle. In our case, the searching time of the exhaustive detection—a function of the number of vertices and arcs in the graph—can be greatly reduced by the following two observations.

(i) Ignore the feedback arcs of the DO-loops since the corresponding cycles have already been detected.

Vertex Number

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
d_1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
d_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
d_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
.																								

Partial Embeddedness matrix E^+

Cycles

Cycle	1	2	3	4	. . .
1	0	0	0	0	. . .
2	0	0	1	0	. . .
3	0	0	0	0	. . .
4	.				

Figure 3—Array D (cycle vertex membership) and matrix E^+ (cycle embeddedness) after DO-loop processing of the example program

TABLE I—List of Origin-Destination statements and I. S.'s

Origin-Destination	Origin-Destination
2, 3	1, 2
2, 4	1, 3
2, 20	1, 10
3, 24	2, 10
7, 6	3, 3
7, 8	3, 4
9, 8	4, 4
9, 10	4, 5
11, 10	6, 3
11, 12	7, 7
12, 5	8, 3
16, 15	8, 9
18, 5	9, 10
18, 19	9, 2
19, 20	5, 5
19, 3	5, 6

(ii) In FORTRAN only a subclass X of statements ($X \subset A$) can be origin statements of cycles. This subclass X corresponds to statement types*:

GO TO K
GO TO L, ($K_1, K_2, \ldots, K_p$)
GO TO ($K_1, K_2, \ldots, K_p$), L
IF (E) K_1, K_2, K_3
IF (B) S where S is a GO TO statement

Only statements of type belonging to X can be starting points of cycles in $G(W, U)$, and they are necessarily terminal statements of I.S.'s. Thus we are going to reduce our cycle search by looking first for cycles in $H(I, Y)$ and then transform these into cycles of $G(W, U)$. This phase is going to be processed in three stages.

Stage 1. Reduce domain of search of elementary cycles of I.S.'s

A list of paired origin-destination statements, where the origin statements are the terminal statements of the I.S.'s and the destination statements are those referred to by the labels K_i, is built during the scanning of the source program Z. The forward reference labelling problem is dealt with in the same manner as the compiler must do for generating code. The corresponding list of origin-destination I.S.'s is also constructed (cf. Table I for our example).

* We restrict ourselves to the analysis of main programs or subroutines, that is we do not include CALL statements as statements which imply transfer of control.

I. S. Number

I. S. Number

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	1
3	0	1	1	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1	1
5	0	1	1	1	1	1	1	1	1	1
6	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1	1
9	0	1	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0

Starter I. S. Number

3

4

5

6

7

8

Figure 4—Matrix M* and domain of starter I.S.'s

The domain of possible starter nodes for the cycle detection algorithm could be further reduced as follows. Let (I_k, I_j) be an origin-destination pair of I.S.'s. They will be part of a same cycle if and only if $m_{kj}^* = m_{jk}^* = 1$ in the reflexive transitive closure M^* of the connectivity matrix M. If these relations do not hold the origin-destination pair can be deleted. However, it is debatable if the gain incurred by this deletion surpasses the cost of generating M^*, a process of order greater than n^2.

Figure 4 shows M^* and the reduced set of I.S. starting vertices.

Stage 2. Find elementary cycles of I.S.'s

The algorithm already referred to[12] is used with the short-cut of skipping those vertices which are not part of the reduced set of possible starting points. Figure 5 lists these cycles.

Stage 3. Conversion of I.S. cycles to original graph cycles (G-cycles)

Each cycle of I.S.'s must be converted into cycles of the original graph (G-cycles). It is of course possible that a single I.S. cycle may yield several G-cycles. For example, the I.S.:

	Corresponding vertex number
10 A=B	1
20 C=D	2
IF (E) 10,20,30	3

which has one I.S. cycle, namely itself, yields two G-cycles $\{1, 2, 3\}$ and $\{2, 3\}$.

The conversion algorithm may be summarized as follows: Let $C_i = I_\alpha, I_\beta, \ldots, I\sigma$ be an I.S. cycle. Recalling the definition of an I.S. one can write $I_\alpha = S_{\alpha,1}, S_{\alpha,2}, \ldots, S_{\alpha,m\alpha}$. All $S_{\alpha,m\alpha}$ are part of all G-cycles generated by C_i. Let $(S_{\alpha,m\alpha}, S_{\beta,j})$ be a paired origin-destination statement leading from I_α to I_β. Now the sets d_i corresponding to G-cycles are obtained as follows:

1. Initialization step. $d_i^0 = \phi$ (the empty set).
2. Let α be the index of the first I.S. in C_i and β the index of the next I.S. in sequence (if there is only one I.S. $\beta = \alpha$).

$$d_i^\alpha = d_i^0 \cup \{S_{\alpha,m\alpha}\}$$

C_1 = (3)

C_2 = (3,4,5,6)

C_3 = (3,4,5,6,7,8)

C_4 = (4)

C_5 = (5)

C_6 = (7)

Figure 5—List of cycles of instruction sequences

3. For each origin-destination pair $(S_{\alpha,m\alpha}, S_{\beta,j})$ do:
   For each d_i^{α} do:

$$d_k^{\beta} = d_i^{\alpha} \cup \{S_{\beta,j}, \ldots, S_{\beta,m\beta}\}$$

4. $\alpha = \beta$; $\beta = \gamma$ (set to index of next I.S.). (If $\alpha = \sigma$, β is set to the index of the first I.S.)

If α = index of the first I.S. then stop; the d_i's are the last $d_k\beta$ generated; else go to step 3. Figure 6 shows the set of G-cycles generated in the Boolean notation adopted in phase 2.

Phase 4

Complete and reduce the embeddedness matrix E^+

We now want to complete E^+ which was defined and partially built during phase 2. If d_i and d_j are two G-cycles in Boolean representation, then let $f_{ij} = d_i \cap d_j$. Elements of E^+ will now be defined by:

—If $f_{ij} = d_i$ then $e_{ji}^+ = 1$, $e_{ij}^+ = 0$ (d_i embedded in d_j)

—If $f_{ij} = d_j$ then $e_{ji}^+ = 0$, $e_{ij}^+ = 1$ (d_j embedded in d_i)

—If $f_{ij} \neq d_i \neq d_j$ then $e_{ji}^+ = e_{ij}^+ = 0$.

Since the embeddedness of the DO-loops between themselves have already been recorded, these tests are done only for the pairs of cycles i, j where $1 \leq i \leq n$ (n number of cycles), $k < j \leq n$ (k number of DO-loops).

Now the E^+ matrix can be considered as the precedence matrix of an acyclic directed graph where the embeddedness property is the precedence relation, i.e., the vertex representing d_i is a successor of the one represented by d_j if $e_{ji}^+ = 1$. In the sequel we shall use strict embeddedness which is defined as: d_i is strictly embedded in d_j if it is embedded in d_j and there exists no d_k such that d_i is embedded in d_k and d_k is embedded in d_j. In order to find a representation of strict embeddedness one has to find the minimum connectivity matrix E such that $E^+ = \bigcup_{i=1}^{\infty} E^i$. Solutions to this problem can be found in Reference 3, 11. Figure 7 shows E for our example program.

This construction completes our modelling process. We now describe some possible applications.

Vertex Number

Cycle Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
d_1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
d_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
d_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
d_4	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d_5	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
d_6	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
d_7	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d_8	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
d_9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Figure 6—Array D (cycle node membership) for all cycles of example program

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	1	0	0	1	1	0
6	1	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

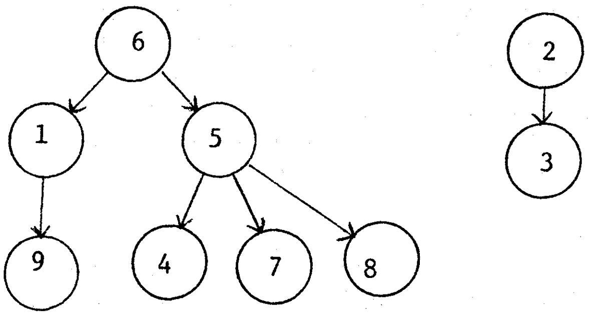


Figure 7—Embeddedness matrix E and associated graph

TABLE II—Matrix S

I	MAXLEVL	USED	NAME	LEVL	DEGREE	FATHER
1	3	0	1	2	1	6
2	2	0	2	1	1	0
3	2	0	3	2	0	2
4	3	0	4	3	0	5
5	3	0	5	2	3	6
6	3	0	6	1	2	0
7	3	0	7	3	0	5
8	3	0	8	3	0	5
9	3	0	9	3	0	1

APPLICATION OF THE MODEL TO PAGING AND CACHE MEMORY SYSTEMS

The model can be used as an aid in packing code in pages in such a way that the number of interpage transitions due to instructions is reduced in comparison with the compiler's usual memory allocation technique. Other attempts at solving this problem range from a macroscopic approach[9] to the analysis of generated code.[5] We will follow a middle of the road route similar in its degree of detail to other studies.[7,13] We do not try in any way to achieve an optimal packing such as in Reference 14 which not only requires information that the model cannot deliver such as the cost of page faulting[15] but which would also take too much computing at compilation time. Our goal is to reduce the number of page transitions through easily implementable algorithms with the understanding that some gross assumptions may have to be taken. The basic philosophy is to pack first those statements which are in the most nested cycles. This was the purpose of obtaining the matrix E.

We need a few additional definitions. Extending the usual relations found in trees, we define the roots of an acyclic directed graph to be those vertices which have no predecessors. The immediate successors of a vertex will be its sons, its immediate predecessors will be its fathers, and sons of the same father will be brothers. Note that this last relation is transitive only in the case of a tree. The level l_i of a vertex w_i is the length of the longest path from a root to w_i. The maximum level ml_i is defined as $\max(l_i, l_j, \ldots, l_k)$ where $w_j, \ldots, w_k$ are the successors of w_i. These levels l_i and ml_i can be computed directly from E^+ or by performing a topological sort on the graph.

Traversal of the graph

Our first objective is to traverse the graph in an order reflective of the nestedness of cycles. A sort on levels l_i, which satisfies this criterion, is rejected because at the same time we want to keep as much as possible the image of the structure of the source program. To facilitate the traversal, the graph is going to be modified into a forest. If in the original graph the vertex w_i had more than one father, the one selected to stay is the one with greatest maximum level. That is for all columns i which have more than one bit set to 1 in matrix E, let j be the index such that $e_{ji}=e_{ki}=e_{li}=\cdots=e_{pi}=1$ and $\mathrm{ml}_j=\max(ml_j, ml_k, ml_l, \ldots, ml_p)$. Then set all bits of column i to 0, except for e_{ji} which is set to 1. After such a transformation is completed the levels and maximum levels have to be recomputed. The forest can now be represented in a tabular form[4] (cf. Table II) which aids in the traversal of the forest. This traversal, a variation on post-order is defined recursively by:

1. Order the sons of the root r by descending maximum levels in SONS$_r$, say $(r_1, r_2, \ldots, r_k)$.
2. Traverse the sub trees of roots $r_1, r_2, \ldots, r_k$ in that order. When a subtree is composed only of a single vertex, its root, the root is then traversed.

In the tabular form, the field USED is a flag showing those vertices which have already been traversed, DEGREE is a counter for the sons which have not yet been traversed and the other fields are self-explanatory. For our example, the algorithm yields {9147856} and {32} for the traversal of the two trees.

During the traversal of the forest, the volume of each individual cycle is computed as follows: Let d_i be a cycle, vd_i its volume, $(w_{i1}, w_{i2}, \ldots, w_{ik})$ its members and $(v_{i1}, v_{i2}, \ldots, v_{ik})$ be the volumes of the w_{ij}. Set d is the set of vertices which have yet to be traversed (initially $d=W$) and d_i is the cycle currently traversed. Then if $d_i'=d_i\cap d$,

$$vd_i=\sum_{w_{ij}\epsilon d_i} v_{ij}$$

and one suppresses from d all vertices belonging to d_i.

Assign vertices to pages

Given *i*) a page size p, ii) the order in which cycles are traversed per the above algorithm, say $T=\{d_{i1}, d_{i2}, \ldots, d_{ik}\}$ and iii) their volumes $VD=\{vd_{i1}, vd_{i2}, \ldots, vd_{ik}\}$ an algorithm to assign vertices to pages, a direct descendant of Lowe's,[7] is summarized here. The algorithm accesses the cycles in T consecutively left to right, assigning as many cycles d_{ij} (each cycle in its entirety) to a single page as the memory volumes vd_{ij} permit. By a cycle in its entirety is meant that a cycle in its vertex assignment is not permitted to run off the end of one page and have following vertices assigned to a subsequent page except when the cycle is the initial (and hence sole) cycle assigned to the page. Whether the cycle is assigned to a page as in the page over-run case or in aggregate vertex by vertex as in the former case, the assignment is reflected by a merger of graph vertices in connectivity matrix C and by the merger of memory volumes in the volume vector V (cf. phase 2 of the second section).

The algorithm consists of three parts. The first part deals with initialization for assigning a new cycle from T to pages. The second considers assignments to pages when a cycle is known to fit entirely within a page and is therefore not assigned to the page on a vertex by vertex basis. The third part considers the case when a cycle exceeds the length of a page and has its vertices assigned to the initial and following pages on a vertex by vertex basis. The algorithm terminates when the cycles in T have been exhausted.

A step by step description of the algorithm follows.

0. Set the current page size cp to 0. Set $h=0$.

A. [*Initialization*].

1. If $h=k$ (i.e., vector T completely traversed) then end.
2. Let h be the index of the next (initially first) cycle in vector T and d_j be the Boolean representation of this cycle.

$$h=h+1; \qquad j=T(h).$$

3. Let n be an index varying from 1 to m (the number of vertices in G). Set $n=0$.
4. If $cp=0$ go to step 7.
5. If $cp+vd_h \leq p$ (i.e., if the current cycle will not go over a page boundary) go to step 11.
6. Set $cp=0$ (Start a new page).
7. $n=n+1$.
8. If $d_{jn}=0$ (i.e., vertex n does not belong to cycle j) go to step 7.

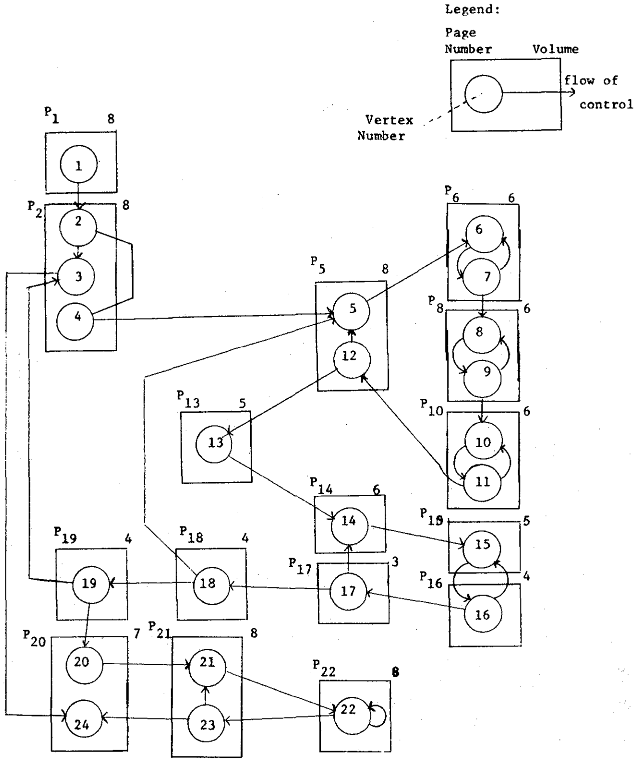


Figure 8—Vertex assignment for example program after first packing algorithm

9. $q=n$ (q temporary variable for last vertex traversed).
10. If $vd_h>p$ (if cycle does not fit entirely in a page) go to step 16 (part C).

B. [*The cycle fits in* (*remainder of*) *page*].

11. $cp=cp+vd_h$; $v_q=cp$ (This is part of the merging process).
12. $n=n+1$.
13. If $d_{jn}=0$ go to step 15.
14. $v_n=0$. Merge w_n into w_q (for an explanation of merging see below).
15. If $n=m$ go to step 1 (i.e., to part A) else go to step 12.

C. [*The cycle overflows a page*].

16. $cp=v_n$.
17. $n=n+1$.
18. If $d_{jn}=0$ go to step 22.
19. If $v_q+v_n>p$ go to step 21.
20. Merge w_n into w_q and v_n into v_q; $cp=cp+v_n$; Go to step 22.
21. $q=n$; $cp=v_n$.
22. If $n=m$ go to step 1 else go to step 17.

TABLE III—Volume Vector for Example Program

Vertex	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Volume	8	4	2	2	3	3	3	3	3	3	3	5	5	6	5	4	3	4	4	5	5	8	3	2

The merging process is defined by:

—To merge w_n into w_q replace qth row of C by the logical sum of itself and the nth row of C, and replace the qth column of C by the logical sum of itself and the nth column of C. Set the nth column and row of C to all 0's.

—To merge v_n into v_q replace v_q by v_q+v_n in the memory volume vector V and set v_n to 0.

The remaining task is to assign those nodes which are not members of cycles. It is done by following part C above, after first initializing both q and n to 1.

Figure 8 shows the result of packing for $p=8$ (a very small page size only for illustration purpose) and the volume vector of Table III. Although this algorithm is crude—and we see below how to improve it—it is worthwhile to compare its outcome with what would have been obtained by usual compiling techniques. Figure 9 shows how the code would be allocated in a straightforward manner with vertices allowed to run over pages.

As can be seen the total number of pages required for the generated code is smaller in the usual compiler's case (7 instead of 10). However, the goal of the packing algorithm is to improve the residency of the pages, i.e., the locality of the code. In general the number of data pages needed for efficient execution is much greater than the number of code pages; the packing algorithm thus helps in leaving more main memory space to data during long bursts of execution. For example, loops (w_6, w_7) (w_{10}, w_{11}) and (w_{22}) occupy one page instead of two. Moreover, as stated earlier, the packing algorithm can be improved. A first clean-up pass could be implemented as follows:

If after the merging process, $C_{ji}=1$ (i.e., pages i and j are connected) and $v_i+v_j \leq p$ (the sum of their memory volumes is less than a page), then merge w_i into w_j and v_i into v_j.

In our example, pages 16 and 17 would be merged, as well as pages 18 and 19, thus leading to a definite improvement in the second part of the main cycle.

A more sophisticated clean-up pass would be first to "squeeze" those elementary cycles which are leaves of the forest, by allowing vertices to run over pages. (In our example w_{16} would be in pages 15 and 16.) Then the clean-up pass defined above would be invoked. Figure 10 shows the results of the improved algorithm. The main loop requires now only 8 pages; the number of interpage transitions is diminished, and the locality of the code is improved.

EXPECTED NUMBER OF EXECUTIONS OF SOURCE STATEMENTS

The knowledge of the expected number of times a given source statement is going to be executed can be an important factor in the optimization of high-level source programs.[6,10] It allows replacement of compiler-generated code by a tighter hand-coded version in the

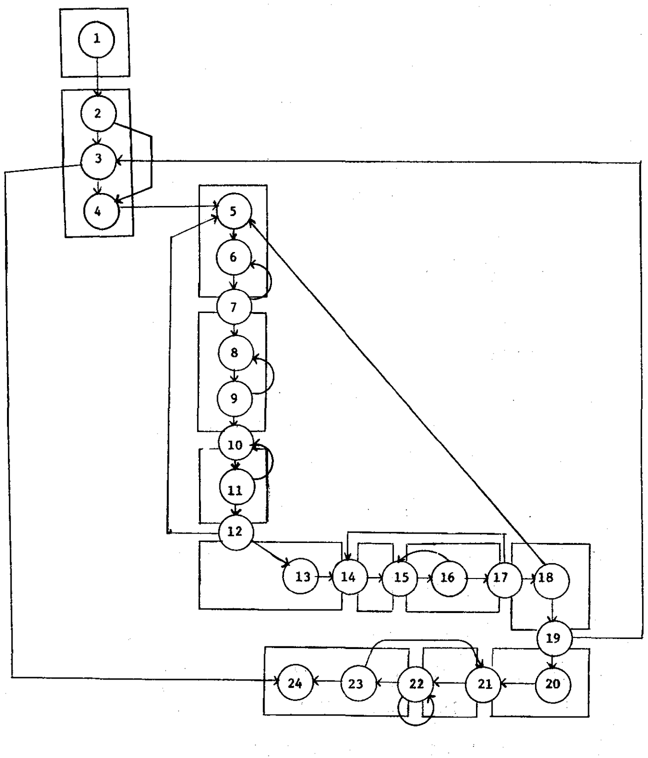


Figure 9—Usual memory allocation by a compiler (all pages have volume 8)

event of an often-repeated statement. In the context of the previous section, the traversal of the tree could be monitored by the maximum expected number of executions, instead of being directed by maximum levels, thus leading to a better residency of pages of code.

In this section we show how these numbers can be estimated when i) the probabilities q_i^j of traversing the arcs (w_i, w_j) corresponding to paired origin-destination statements are known, and ii) when the mean number of times the feedback arc of a DO-loop is set up to be traversed is known. These parameters of the source program will have to be "guessed" or instrumented.[10] In the former case they represent a first pass at the analysis of the program and only well-known sections of it should be optimized. In the latter, they correspond to an "a posteriori" optimization of a production run. In both cases, we assume that we have a Markovian process, i.e., the q_i^j stay invariant each time w_i is reached.

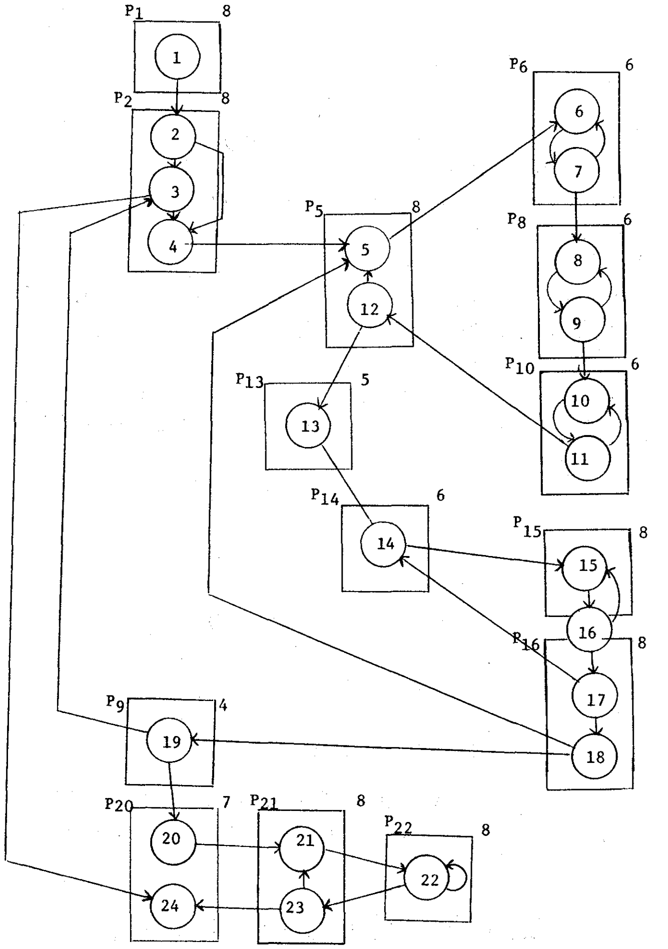


Figure 10—Parking after "squeeze" and "clean-up" passes

Traversal of the directed graph

The method we outline is an outgrowth of studies related to the prediction of performance of multiprocessors.[2,8] Let f_i be the expected number of executions of vertex w_i, p_i its probability of being reached from the initial vertex w_1 when all feedback arcs of the graph have been removed. In the case of a stochastic cycle, i.e., one which is not a DO-loop, this feedback arc is the one corresponding to the last paired origin-destination statement encountered when transforming the I.S. cycles to G-cycles, and of probability not equal to 1. After normalizing the probabilities of arcs incident out of vertices which are origins of feedback arcs, one computes the p_i as:

(1) $p_i = \sum_{j=1}^{k} p_j q_j^i$ where w_j $(j=1, \ldots, k)$ are the immediate predecessors of w_i. At initialization all f_i are set to 1.

The traversal of the directed graph represented by the original embeddedness matrix E is performed as follows.

1. Let $d_1, d_2, \ldots, d_k$ be the leaves of the graph. Partition these leaves into classes $E_1=\{d_1\}$, $E_2=\{d_2\}, \ldots, E_k=\{d_k\}$.
2. If two classes E_i and E_j are such that there exists $d_i \in E_i$ and $d_j \in E_j$ such that $d_i \cap d_j \neq \Phi$, let $E_i = E_i \cup E_j$ and suppress E_j.
3. Repeat step 2 until no suppression occurs.
4. Compute the f_i (up to that point) of each vertex member of a cycle belonging to one of the E_i left after step 3, as explained below.
5. Suppress from E all the leaves and repeat steps 1 to 4 until all cycles have been treated.

Steps 1 to 3 are facilitated by the use of the tabular representation of the graph. They involve mainly the partition of cycles into classes with the equivalence relation being "to have a common father with at least one member of the class."

Computation of f_i's corresponding to leaf cycles

Let $E_k = \{d_{i1}, d_{i2}, \ldots, d_{ik}\}$ be a class as defined above. We shall give expressions for the f_i of vertex members of d_{ij} in supposing first that the cycles are at their maximum depth of embeddedness.

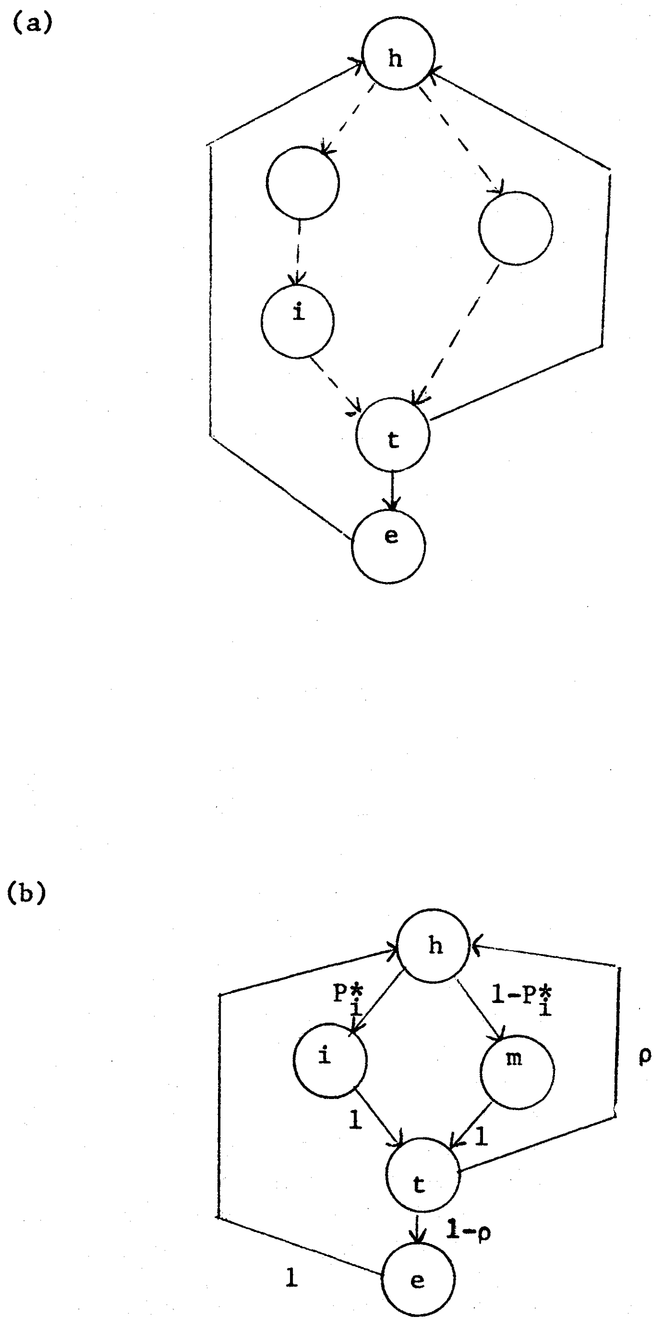


Figure 11—Regular Markov chain (subcase 1 of case 1)

Case 1. E_k has a unique member $d_{i1}=\{w_h, \ldots, w_i, \ldots, w_t\}$ where (w_t, w_h) is the feedback arc.

Let p_i^* be computed as in (1) in considering only the subgraph corresponding to d_{i1} and $p_h=1$. We note $\Pr[w_i(j)]$ as the probability of executing $w_i\, j$ and j times only.

Subcase 1. There is no branching out of d_{i1}, that is there exists no arc (w_i, w_i') such that $w_i \in d_{i1}-\{w_t\}$ and $w_i' \notin d_{i1}$.

Let $(N-1)$ be the mean number of times the DO-loop has been set up for and ρ the probability of the feedback arc of a stochastic cycle. For a DO-loop:

$$f_i=\sum_{j=1}^{N} j \Pr[w_i(j)]$$

and

$$f_h=N.$$

But

$$\Pr[w_i(j)]=\binom{N}{j} p_i^{*j}(1-p_i^*)^{N-j}.$$

Hence:

$$f_i=\sum_{j=1}^{N} j \binom{N}{j} p_i^{*j}(1-p_i^*)^{N-j}$$

$$=Np_i^* \sum_{j=1}^{N} \binom{N-1}{j-1} p_i^{*j-1}(1-p_i^*)^{(N-1)-(j-1)}$$

or

$$f_i=Np_i^*=f_hp_i^*.$$

For a stochastic cycle: the process can be considered as a regular Markov chain as shown in Figure 11.a. by introducing an extra arc of probability 1. Now if P is the regular transition matrix, $(\Pi_h, \ldots, \Pi_i, \ldots, \Pi_t, \Pi_e)$ the limiting probability vector, it is known that the mean first passage time to return to a given state is $m_{ii}=1/\Pi_i$. In our case, $m_{ee}=1/\Pi_{ee}$ represents the mean number of steps, and $f_i=\Pi_i/\Pi_e$.

If one is interested only in f_h, f_i, f_t, the graph is equivalent to the one in Figure 11.b. whose transition matrix is:

$$\begin{vmatrix} 0 & p_i^* & 1-p_i^* & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \rho & 0 & 0 & 0 & 1-\rho \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix}$$

and since $\Pi_i=p_i^*\Pi_h$, then $fi=p_i^*f_h$. Now f_h could be computed by the ratio Π_h/Π_e but directly one can see that:

$$f_h=\sum_{j=1}^{\infty} j \Pr[w_h(j)]=\sum_{j=1}^{\infty} j\rho^{(j-1)}(1-\rho)=\frac{1}{1-\rho}$$

Subcase 2. There is some branching out of total probability $1-Q$ where $Q=p_t^*$. In the DO-loop case:

$$\Pr[w_h(j)]=Q^{j-1}(1-Q)\ (j=1,\ldots,N-1);$$

$$\Pr[w_h(N)]=Q^N.$$

Hence

$$f_h=\sum_{j=1}^{N} jQ^{j-1}(1-Q)+NQ^{N-1}=\frac{1-Q^N}{1-Q}$$

Now

$$\Pr[w_i(k)]=\sum_{j=k}^{N}\Pr[w_h(j)]\binom{j}{k}p_i^{*k}(1-p_i^*)^{j-k}\qquad (k=1,\ldots,N)$$

and

$$f_i=\sum_{k=1}^{N}k\Pr[w_i(k)]$$

$$=\sum_{k=1}^{N}\Pr[w_h(k)]\sum_{j=1}^{k}j\binom{x}{j}p_i^{*j}(1-p_i^*)^{k-j}$$

$$=p_i^*\sum_{k=1}^{N}k\Pr[w_h(k)]\cdot$$

$$\sum_{j=1}^{k}\binom{x-1}{j-1}p_i^{*j-1}(1-p_i^*)^{(k-1)-(j-1)}$$

$$=p_i^*\sum_{k=1}^{N}k\Pr[w_h(k)]=p_i^*f_h.$$

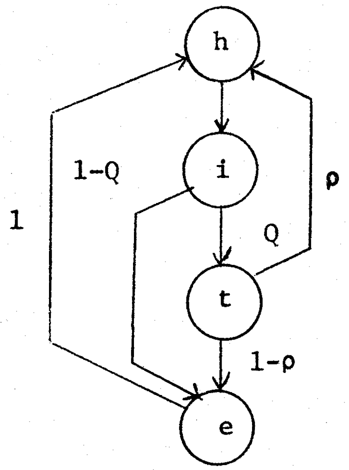


Figure 12—Regular Markov chain (subcase 2 of case 1)

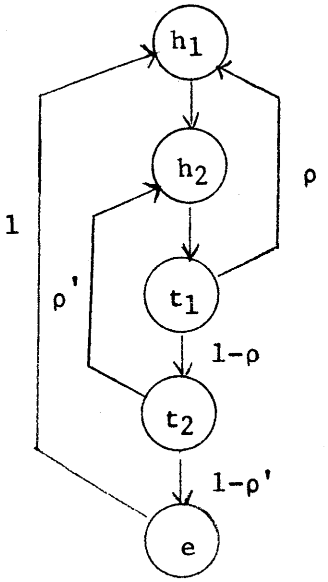


Figure 13—Regular Markov chain (subcase 1 of case 2)

In the stochastic case, the computation of f_h corresponds to the Markov chain of Figure 12 of transition matrix

$$\begin{vmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & Q & 1-Q \\ \rho & 0 & 0 & 1-\rho \\ 1 & 0 & 0 & 0 \end{vmatrix}$$

which yields

$$f_h=\frac{1}{1-Q\rho}$$

Analysis similar to subcase 1 would yield $f_i=p_i^*f_h$. In summary for this first case, the computation of the f_i involve only the computation of the p_i^* and of the f_h as given above.

Case 2. E_k has more than one member.

Subcase 1. All cycles are stochastic.

Again we have recourse to the Markov chain approach. Of interest are the f_i of the heads and tails of the cycles. A typical example is shown in Figure 12 of transition matrix;

$$\begin{vmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \rho & 0 & 0 & 1-\rho & 0 \\ 0 & \rho' & 0 & 0 & 1-\rho' \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix}$$

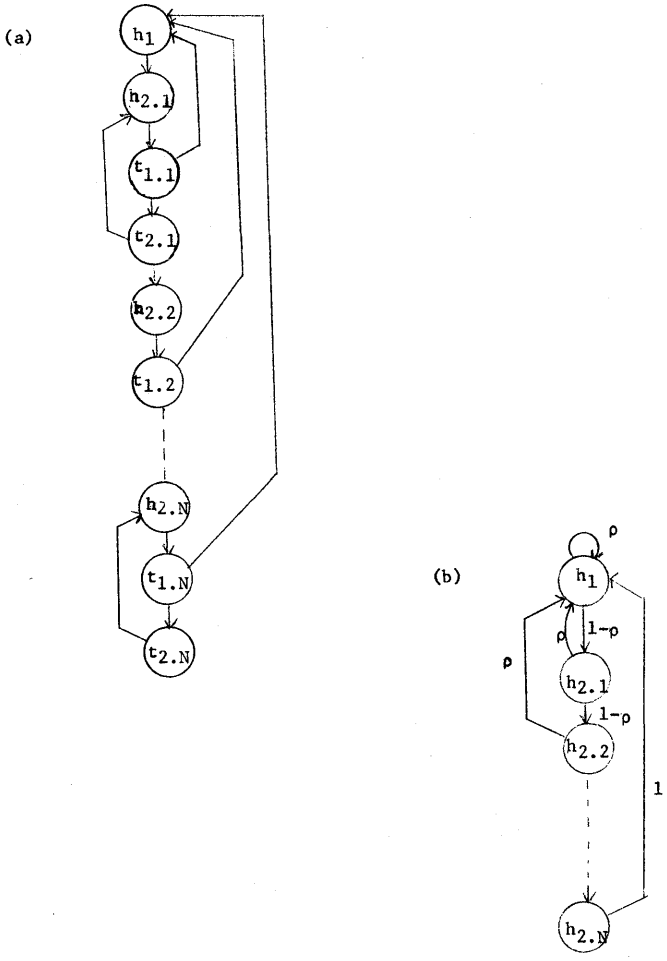


Figure 14—Subcase 2 of case 2

which yields

$$f_{h1}=1+\frac{p}{(1-p)(1-p')},$$

$$f_{h2}=f_{t1}=\frac{1}{(1-p)(1-p')}$$

and

$$f_{t2}=\frac{1}{1-p'}$$

Subcase 2. One member of E_k is a DO-loop (there can be no more than one due to the Syntax of FORTRAN).

Consider again Figure 13 but suppose now that the second cycle is a DO-loop. It could be possible by using methods of renewal theory to determine the f_i in this case. However we can still use the Markov chain approach. To do so an N-folding of the structure must be performed as shown in Figure 14.a. To obtain f_{h2} and f_{t2} a reduction as in Figure 14.b. is performed giving a transition matrix

$$\left|\begin{matrix} p & 1-p & 0 & \cdots & 0 \\ p & 0 & 1-p & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ p & 0 & 0 & \cdots & 1-p \\ 1 & 0 & 0 & \cdots & 0 \end{matrix}\right|$$

which yields

$$f_{h1}=\frac{1}{(1-p)^N},$$

$$f_{h2}=\sum_{i=1}^{N} f_{h2\cdot i}=\sum_{i=1}^{N}\frac{1}{(1-p)^i}=\frac{1-(1-p)^N}{p(1-p)^N}$$

and $f_{t1}=f_{h2}$, $f_{t2}=(1-p)f_{h2}$ as in subcase 1.

If for both cases 1 and 2, the cycles are not embedded in any other cycle, the f_i are multiplied by p_h the probability of reaching the first head. If they are embedded in other cycles, the f_i computed at the next level will be multiplied by those just obtained until all cycles have been treated.

CONCLUSION

In this paper we have presented a model of Fortran source programs with particular emphasis on the cyclic structure. Applications of the model to segmentation and optimization have been introduced. Algorithms needed to extract information from the high-level language in order to build the model have been given. A packing algorithm, whose goal is to reduce the number of interpage transitions in a paging or cache memory system, has been presented along with its application to an example program. The cyclic structure of the program has also been used to compute the expected number of executions of each statement of the source program.

The results of the packing algorithm, which is meant to increase the locality of the generated code, could also be used in paging mechanisms with anticipatory control. The expected numbers of executions could serve in the packing of data into pages, the data with high correlation being allocated in the same pages. This latter problem needs more investigation and it is in this direction that extensions of the model and new algorithms should prove useful.

REFERENCES

1 F E ALLEN
Program optimization
Ann Rev in Aut Prog 5: pp 239-302 1969

2 J L BAER
Graph models of computations in computer systems
PhD Dissert UCLA 1968

3 J L BAER
Matrice de connexion minimale d'une matrice de précédence donnée
RIRO 16: pp 65-73 1969

4 R T CAUGHEY
Automatic segmentation of FORTRAN programs from cyclic structure analysis
MS Thesis Univ of Washington 1971

5 F H DEARNLEY G B NEWELL
Automatic segmentation of programs for a two-level store computer
The Computer Journal 7 3 pp 185-187 May 1968

6 D E KNUTH
An empirical study of FORTRAN programs
C Sci Depart Report CS-186 Stanford Univ 1970

7 T C LOWE
Automatic segmentation of cyclic structures based on connectivity and processor timing
Comm ACM 13 1 pp 3-9 Jan 1970

8 D F MARTIN G ESTRIN
Models of computational systems cyclic to acyclic graph transformations
Trans IEEE EC-16 pp 70-79 Feb 1967

9 C V RAMAMOORTHY
The analytical design of a dynamic look-ahead and program segmenting scheme for multiprogrammed computers
Proc ACM 21st Nat Conf Thompson Book Co pp 229-239 1966

10 E C RUSSELL G ESTRIN
Measurement based automatic analysis of FORTRAN programs
Spring Joint Conf Proc 34 pp 723-732 1969

11 J M S SIMOES-PEREIRA
On the Boolean Matrix Equation $M' = \bigcup_{i=1}^{\infty} M^i$
Journal ACM 12 3 pp 376-382 July 1965

12 J C TIERNAN
An efficient search algorithm to find the elementary circuits of a graph
Comm ACM 13 12 pp 722-726 Dec 1970

13 E W VERHOEF
Automatic program segmentation based on Boolean connectivity
Spring Joint Comp Conf Proc 38 pp 491-495 1971

14 B W KERNIGHAM
Optimal sequential partitions of graphs
Journal ACM 18 1 pp 34-40 Jan 1971

15 J KRAL
One way of estimating frequencies of jumps in programs
Comm ACM 11 7 pp 475-479 July 1968

APPENDIX I

Vertices and arcs generated by the graph modeling process.

Notation: w_i current vertex number; w_{i+1} next (sequential) vertex number; w_n vertex corresponding to source statement of label n; w_z terminal vertex.

STATEMENT	VERTEX GENERATED	ARCS GENERATED
GO TO n	w_i	(w_i,w_n)
GO TO $(n_1,n_2,\ldots,n_k)$,j	w_i	$(w_i,w_{n1}),(w_i,w_{n2}),\ldots,(w_i,w_{nk})$
GO TO j,$(n_1,n_2,\ldots,n_k)$	w_i	$(w_i,w_{n1}),(w_i,w_{n2}),\ldots,(w_i,w_{nk})$
IF (e) n_1,n_2,n_3	w_i	$(w_i,w_{n1}),(w_i,w_{n2}),(w_i,w_{n3})$
IF (e) S		
(a) If S is not a GO TO RETURN or STOP	w_i,w_{i+1}	$(w_i,w_{i+1}),(w_i,w_{i+2})$
(b) If S is a GO TO RETURN or STOP	w_i	(w_i,w_{i+1})
DO n I = M1,M2,M3	w_i	$(w_i,w_{i+1}),(w_n,w_{i+1})$
CONTINUE	w_i	(w_i,w_{i+1})
PAUSE	w_i	(w_i,w_{i+1})
RETURN	w_i	(w_i,w_z)
STOP	w_i	(w_i,w_z)
READ(a,b,END$=n_1$,ERR$=n_2$)list	w_i	$(w_i,w_{i+1}),(w_i,w_{n1}),(w_i,w_{n2})$
WRITE(a,b)list	w_i	(w_i,w_{i+1})
ENDFILE	w_i	(w_i,w_{i+1})
REWIND	w_i	
REWIND	w_i	(w_i,w_{i+1})
BACKSPACE	w_i	(w_i,w_{i+1})
Arith. Stats.	w_i	(w_i,w_{i+1})
END	w_z	

The interplay of computer science and large scale scientific calculations

by KENT K. CURTIS

National Science Foundation
Washington, D.C.

The history of computing has been characterized by the statement that what the manufacturer makes, the customer takes. To the extent that this is true, it is not only an interesting and perceptive comment on history but also a testimonial to the remarkable nature of the sequential, stored program, electronic computer. If that invention had not fit so well from the beginning to a wide variety of interesting problems, the development of hardware and software systems could not have proceeded as independently of scientific motivation as it did. Of course, scientific requirements have influenced systems design, but the prominent parameters such as speed or memory size could be understood without detailed knowledge of scientific problems or programs and the parameters which might depend upon the structure of programs could be ignored.

It is fortunate that that was true. There were important problems to solve in physics, chemistry, meteorology, etc. We could work on those without delay and let systems design take care of itself, with *ad hoc* software adjustments when things became intolerable. Computers worked so well there was no apparent need for guidance by scientists of system design. Better design might make an incremental difference but that increment was not sorely missed and barely worth an incremental effort.

Meanwhile, computer development followed an internal dynamic of its own. It was influenced by the fact that scientific requirements provided a ready market for computers which were big and fast, but the general purpose computer was a useful paradigm and rapid advancement of electronics technology kept many problems at bay. Computer engineers and businessmen could do their own thing in their own way, with unusual freedom. The computers they built have reflected principally their own genius in taking advantage of technological opportunity, and that, happily, has been sufficient.

In making these comments, let me note that they refer to the machines which have until now dominated scientific calculation. Other designs which attempt to reflect some additional measure of understanding of scientific calculations have been considered and some are of substantial current interest. Pipe-line systems and the ILLIAC IV, are among those which come to mind, but none of them has yet had opportunity to prove its value. Even they, however, are the result of adding only one more global observation to the design criteria, the observation that many scientific calculations have the common property that they involve the redundant use of identical operation sequences on ordered sets of data. Hence, their appearance and history does not vitiate the force of the earlier observation. On the contrary, they extend its force to the near future.

They do, however, raise several interesting points. First, the principal ideas of architecture underlying each of the machines which are now being physically realized are approximately ten years old. Still, these machines are only now being delivered and experience suggests that it will be some time yet until the systems will have been stabilized sufficiently and used sufficiently to be able to make a definitive evaluation of them from experience. This is a long gestation period. It suggests that we should not expect breakthroughs in large scale scientific computing due to new ideas in machine architecture in a short time scale. The machines which are now coming to life may be substantial contributors to scientific computing ten years from now, if they prove out, but it is unlikely that machines which are still on the drawing board will contribute until later.

It also suggests that unless we believe we already have discovered the only important concepts in machine organization (which is possible but would be most remarkable, if true) we should be very interested in reducing the time span from idea to realization. Ideas in machine organization spring from an environment of experience to meet a perceived need. They need testing to evaluate them. If the need is really important it will probably persist until an answer is found. But, on the other hand, if the need is important we would rather not wait ten years. This line of thought argues for research in such areas as description languages which can describe complex systems and be translated to a design and fabrication process and it argues for support by the scientific community for such research. I would like to return to this point later.

Second, a "paper machine" has very little weight. At an early stage in the development of the machine which is becoming ILLIAC IV, I spent some time with several people who were active in hydrodynamics and meteorological research to see what basis for estimating performance could be established from analyses of the actual codes which were of interest in those fields. My question was, to what extent does the parallelism which is available in this design offer an effective improvement in performance, considering the detailed structure of the computing algorithms, data management, and I/O processes that are used in these problems? I thought I understood how to do that kind of analysis, at least in part, including modifying the codes for parallel processing, but I did not know the codes. I hoped that someone who knew the codes could be motivated to undertake some analysis. I was impressed with the degree of interest in the machine and the lack of interest in attempting any analysis. I found a uniform response that if the machine existed, they would try it because it seemed intuitively attractive. No one was willing to attempt an analysis of his programs, however, with a view to justifying construction of the machine or influencing its design. In all fairness, they did not know how to do it and neither did I. The problem is harder and more subtle than I thought. The question remains open and, indeed, I still do not know how to find an answer with assurance except by extended experience. Responsible people still hold an honest difference of opinion. Fortunately, the continued development of that machine did not require the active support and involvement of the scientific community it was intended to serve. Technological opportunity supported by intuition and some analysis has been sufficient. The experience suggests, however, that there exists only a weak coupling between the community of natural scientists and the community of computer engineers. It also affirms the need to test new ideas in machine design by actual construction of machines and evaluation in a user environment.

Another interesting point which these machines have emphasized is that more than one schema may be applied to the solution of most problems and different computer architectures may favor different schema. This seems obvious but it required the actual development of parallel and pipe-line machines to stimulate such rethinking of problem formulation and algorithms. Now this offers a rich field of research for numerical analysts and computer scientists, where results may be of direct interest to scientists. I personally believe that this is a profoundly significant development which may portend the appearance of a new degree of influence by computers and computer science in our conceptualization of the scientific problems we attempt to solve. More immediately, however, it means that definitive evaluation of new ideas in computer architecture can only be made after good algorithms for those architectures are devised. Is this backwards? Should not the analysis of algorithms influence design? Is this not another example of weak coupling between communities with allied interests? Fortunately, we are beginning to make progress on the theory of algorithms which offers hope of benefit to both scientific computing and computer design.

Let us return, now, to the remarkable success of sequential computers in large scale scientific computing. If one considers the spectrum of activities which now comprises large scale scientific calculation one can observe a number of common features.

(1) The conceptual theory upon which the computations are based was firmly established before the invention of computers. That theory had been well verified using experiments and calculations which could be and were conducted without computers, providing a solid basis for faith that large investments of effort and money in computing would not be wasted.

 High energy physics might be considered to be one outstanding exception to this since conceptual theory in that field is being developed during the age of computers. If one considers the actual computing being done, however, the distinction disappears. The large scale computing is in analysis of data from experiments which are patterned in their design and analysis after particle physics concepts that were well established before computers were applied. Orders of magnitude less computation is used in theoretical investigation exploring the conceptual understanding of high energy physics. No theory commands sufficient faith to justify a larger investment of time and money.

(2) The conceptual theories are all expressible in well formed mathematical or statistical models.

(3) Based on the conceptual paradigms, it is possible to visualize, without using computers, interesting problems which can be solved with computers.

(4) Such problems can be solved, to a sufficiently good approximation to be interesting, using sequential machines operating for reasonable lengths of time.

(5) These problems enjoy a high national priority giving them access to the large amounts of money and effort which have been required for their study.

If you think about it, this is an interesting set of properties. It suggests the roles which have been played in fact by scientific motivation, technological opportunity, and public support in determining the course of development of large scale scientific computing. It provides a basis for understanding why scientific progress and the development of computer technology could be so vigorous, so symbiotic, and yet so independent. It provides a rationale for the observation that scientific inquiry has adopted the technology provided rather than leading or strongly cooperating in its development. At the same time, it points to certain limits, which have been observed by the course of events, in fact, on the regions of success in scientific computing. If one may have faith in continuity, it suggests some features which can be expected of developments in the near future.

One interesting observation which derives from this set of characteristics is that although computing has profoundly changed the style and methodology of research in some fields, although it has opened new questions for serious consideration, there has been no departure from already established concepts. Computing has not yet altered our paradigms of nature in a fundamental way. This is not surprising—the time constant for developing fundamental physical theory is much longer than the history of computers—but it indicates that we should not expect the short time scales associated with the development of technology to be reflected in any obvious way, if at all, in the rate we revise our fundamental understanding of nature. It also invites the question of whether computing should be expected to ever become an essential component of conceptualization. Are there interesting phenomena which cannot be described within the framework of mathematical or statistical models? Are there important questions to be posed which do not yield to analysis by sequential procedures? What would it mean to include a program operating in time as an essential element in our description of a model, as we now include time-dependent equations? It is an interesting conjecture. Considerations of real-time systems are suggestive.

Before going on, let us examine some possible inferences which may be drawn from the preceding characterization of large scale scientific computing. Certainly it is true that national priorities have strongly influenced the choice of fields of investigation and hence the visible progress. These priorities are changing and efforts in large scale computation in fields other than the physical sciences will receive more encouragement. Fields which have the property that they can take advantage of existing technology, have well posed questions based on well formulated mathematical paradigms, and have a sufficiently large community of interested and talented scientists who are beginning to emerge as partners with the physical sciences in large scale computing. The National Science Foundation is playing a role in this through its interest in the applications of computers to research in the natural sciences, in social sciences, and in applied research. One harbinger of change is the establishment of the Computer Research Center for Economics and Management Science by the National Bureau of Economic Research, Inc. with NSF support. We intend to take advantage of other opportunities as they arise.

If we note, again, that it has taken a decade to develop the new machines which are now appearing and that it will be some time, yet, before they can be adequately evaluated we may guess that the new fields which will join the ranks of large scale scientific computing in the next decade can be discerned now. They are not many. Few disciplines enjoy the advantages which the physical sciences have of almost universally accepted, well established paradigms which adapt well to existing computer technology. Advances in theory might accelerate the entry of one field or another in unexpected ways but advances in technology are unlikely to do so within a ten year time frame. I, myself, am not aware of many well formed problems embedded in widely accepted theory which are only awaiting a technological breakthrough to be solved. Even if they exist, it will take time to translate new ideas in technology into effective research application. The ideas must first be converted into actually realized tools, a task that may take more than ten years itself if we accept prior experience as a guide. The application of these tools must then be developed. Although there is overlap in these activities, a reasonable expectation would be that in ten years time we will be able to guess some still in the future impacts we do not now discern but the fully active members of the large scale computing fraternity which will then exist are already recognizable today as having accepted theories which are describable by mathematical or statistical models amenable to solution by sequential machines. The potential new members are distinguished from the present members primarily by the fact that they are only now adopting the mathematical methodology of the physical sciences and by the fact they they do not yet enjoy high national priority status. Technology notwithstanding, ten years is a short time in the development of human thought. Even quantum theory took much longer to evolve.

Another feature which emerges from considering this set of properties is the prevalence in successful scientific computing of certain mathematical or statistical models. Not only were the conceptual theories well established before computers entered the scene but the models used to describe those concepts and the approaches taken to solve them were also well established. For example, finite difference approximations of partial differential equations. We have always quickly exhausted the capacity of the machines we had to solve those problems in that way but we have not become frustrated because technology advanced rapidly enough to keep the game interesting. We have made substantial advances in discovering faster, more efficient algorithms for carrying out these processes and in understanding convergence and stability but basically we have worked, not only in the same conceptual but also in the same procedural framework that was common with desk calculators. (Monte Carlo techniques might be thought to be an exception. I think not. As with many things, computers made those already established techniques more effective.)

Now a new line of thought is beginning to emerge. I am aware of it in meteorology but it may be present elsewhere. Computers have enabled us to undertake "ab initio" calculations which could never be approached before. (Indeed, some chemists now believe that their theory is so sound and their computing techniques so accurate that they can place as much faith in the results of a calculation of a possible molecule as they can in an experiment. They have some substantial basis, as is evidenced by the story of the noble gas compounds but it reflects an interesting shift in attitude.) This ability has enabled us to undertake prediction with an accuracy never before possible and the long range weather forecasting which has become part of our daily lives is an outstanding result. But, perhaps we are beginning to push the limit of that ability using standard techniques. The predictive power of a given mathematical model rests ultimately upon the completeness and accuracy of the physical data we can provide even if the model is in principle perfect and we have infinite computing power. At some point, decreasing the size of the mesh or increasing the number of layers in a three dimensional atmospheric model multiplies the computational steps required without yielding a commensurate increase in useful results. It has been suggested that in atmospheric circulation problems, computers may be bringing that limit within sight. A couple of weeks may be a limit beyond which the model diverges from reality because of limitations in physics, not computing.

This observation is suggestive. The problem has been recognized before and some computer scientists have studied significance arithmetic schemes to provide continuous knowledge of the significance of quantities throughout a calculation. This work has not yet entered into practical application but interest may revive in it as we find ourselves pushing the limits of data accuracy more frequently. If true, it will offer a rich field for research relevant to computer users.

But the observation is also interesting in another dimension. It would be surprising if a conceptual theory can find mathematical expression in only one model or if that model can be solved in only one way. It would also be surprising if we have already chosen the best model for all situations. In atmospheric circulation, some of the questions one wants to raise may be answerable from a turbulence model which is statistical in nature and leads to a different computational problem. This could be a better way to obtain some results than solving the general field equations.

This phenomena is not new. Scientists have always reformulated models and changed approximations when they were unable to make further progress with one particular line of attack. Instead, the last twenty years have been unusual. Computers have increased the power of certain traditional lines of attack so much that we are only now approaching their limits. As we reach those limits, more attention may be paid once again to alternatives and to the real significance of the numbers we compute.

Finally, this set of characteristics of successful scientific computing can be helpful in anticipating the near term impact of new developments. They suggest that the most likely effect of technology will be to help scientists do better the things they already know how to do, perhaps, even stronger, the things they are already doing. Before discussing specifics here, however, it is necessary to make note of a new current which is beginning to influence the computing environment. While scientists and computer manufacturers each followed their own independent but mutually reenforcing interests, a third community, the community of computer scientists, was gradually emerging. This is the community whose imagination is captured by the intrinsic interest of complexity, systems, and computers and it is largely populated by bright young people who grew up with computers as a background. To the extent that computers have broken tradition, these people have the advantage of growing up with fewer ties to the old traditions. This community is now taking shape and beginning to find things to say which have relevance to both scientific calculation and computer design.

Neither the computer manufacturers nor the computer users have had much interest in computer science. They have been preoccupied with technology and have strongly flavored the definition of the term "computer science" with their own interests. For this reason it may be necessary to define more precisely what I mean. To some degree, one man's science may be another man's technology and neither can live in good health without the other. This is especially true if one looks at the actual occupations of people who call themselves scientists, the things they really do with their time and energy. The term science, however, has a commonly accepted meaning which involves the discovery and formulation of generalizations or principles that can be expressed in models describing phenomena. Computing has focussed attention on phenomena which are as old as man but which have been taken for granted; phenomena concerning information, complexity, and the processes for organizing, describing, and analyzing. Computer science is addressing itself to these phenomena. Its success will have meaning for all who are concerned with computers.

The conduct of computer science research is also interesting. There is reason to believe that it must be, at least in part, essentially complex and involve the concerted effort of research teams. Professor Jack Schwartz, of New York University, has used an analogy which I find fruitful. He notes that mathematics is a solitary occupation which can be compared to diamond mining. One digs through large quantities of dross but in the end one finds a jewel which is small, beautiful, and can be easily admired by all who have eyes to see. Computer science, on the other hand, is comparable to coal mining. One handles comparable or greater quantities of material but none of it can be thrown away. Everything must be preserved for use and the process is intrinsically complex. This does not preclude generalization and the discovery of principles but it does say something about the process of finding them and about the description of them which gives them meaning.

With this preamble, let us return to the consideration of new technology. There are several things which come immediately to mind. They cannot really be separated—each influences the others—but you can construct your own feedback loops.

(1) Machine architecture

This has already been discussed. The new architectural ideas which may affect large scale scientific computation within ten years are already being built in hardware. The long lead times involved make it almost certain that the impact of other ideas will not be realized in that time frame. New contributions inspired by the machines now being built will increase rapidly as those machines actually become available, however, and will influence their use. These will be in the areas of problem formulation, analysis of algorithms, data management schemata, operating systems, languages, and file management. I hope these machines will be available for computer science research. I think they cannot be properly evaluated without it.

(2) Solid state memory

It appears likely that we will soon have memories available which are comparable in cost to present core memories but comparable in speed to logic. This will change one of the parameters of machine design which has been nearly constant throughout the history of computers, the ratio of memory time to logic time, from about 100:1 to about 1:1. It would be most surprising if this does not have some effect. Programming would seem to gain a new degree of freedom. One possible effect may be more effective specialization of machine function without specialization of machine design. Floating point arithmetic might be programmed again, not wired, for example, to give greater flexibility of word size and significance arithmetic may have a better chance for effective implementation.

On the other hand, the forces influencing change are numerous, subtle, and complex. It is very difficult to forecast but fascinating to watch as events unfold. The foregoing statements assume that it is easier to program than to build machines. But one can also imagine ways in which this same memory development, with proper inspiration and motivation, might lead to simplifying the fabrication process. Modular building blocks for machines and systems of machines might then become part of our standard repertoire.

(3) Algorithms

This is an area where computer science will give us an undisputed benefit. Theoretical limits on the speed of algorithms will provide measures against which to compare performance and will inspire improvement. Analysis and classification of algorithms will begin to give formal structure to programs and guides to optimize design of both programs and machines. From this area will come the indispensable tools for understanding the information processes which are possible for us to use in solving problems.

The benefit will be felt, not only in increased performance on our present problems, which may be significant or not, but in providing us with constructive procedures for thinking about the information processes appropriate for our problems. It may seem facetious to suggest that the science of algorithms will usurp the role of calculus and numerical analysis as the primary tools for description and problem solving. It certainly is premature. But I will be interested to see what our attitude is about this at the end of this decade.

(4) Languages

In the short history of large scale scientific computing we have relied primarily upon one language, FORTRAN, and have accepted its limitations in the interest of pursuing the scientific problems at hand. Again, it has been the case that other languages might provide an incremental gain but that has not been worth an incremental effort by scientists to learn, to use, or to develop. Improvements have been made and will continue. Computer scientists are developing automatic code optimization procedures which may help performance and, perhaps, debugging. Monitoring schemes and performance measurement studies are making it possible to do time analysis of programs to increase the efficiency of codes. Factors of two, three, ten, or sometimes fifty improvement result. But with respect to learning languages, the scientific community has been conservative.

Now we may be reaching another kind of limit. Programming is a human effort involving a long chain of people and events before a complete program can be made to work satisfactorily. Programming time and the irreducible error rate at each step of the process places a practical limit on the growth of complexity of programs, generally by making the cost or time required to achieve a certain degree of complexity greater than we can, or want to invest. In spite of large programming investments and outstanding successes we are all aware of these limitations, and have had the experience of thinking of interesting programs which we were unwilling or unable to write. More significantly, some of the programs we have already constructed are maximally complex for our resources. FORTRAN is becoming a restraint that can be felt.

It is clear that this restraint can be relaxed since this is an area in which computer science has made progress since FORTRAN was developed. Iverson's APL, for example, is indicative as is Schwartz' SETL. In still another direction, Hearn's REDUCE is building a record of accomplishment. Within a decade, scientists will be multilingual and have greater freedom for scientific research as a consequence.

Another problem in the general area of languages which computer science is attacking is the problem of transportability of programs from one computing environment to another. Gradually, the structure of programs, compilers, and operating systems is becoming clear. We are learning how to isolate the essentially machine dependent portions of codes, and to bootstrap a system with minimum reprogramming effort. The benefits of this development are already being felt in economy of programming effort when changing machines. Ultimately, it may make an important contribution to scientific communication by making it feasible to exchange codes.

(5) Communications

This may make the most significant change in our daily working environment and seems to have large momentum, with stimulation from many sources. The National Bureau for Economics Research Center, which I mentioned earlier, is being established with remote operation in mind from its inception. So is the CDC-7600 installation at Berkeley for high energy physics. ILLIAC IV, at Ames, will soon follow and it seems likely that other centers for specialized disciplines with planned remote operation will soon appear.

The trend seems natural and inevitable. Research computing involves not only a computing machine but also a corpus of appropriate programs and an operating policy favorable to the work. Finally, and perhaps most importantly, it involves a group of scientists and support staff which can work well together and communicate freely to develop the research methodology. The success of this organization for research has been demonstrated in the national laboratories. It is not surprising that it carries over to computing with communications providing the means.

At the same time, this mode of operation is a departure from the organization of research computing which developed during the fifties and sixties when each institution built the local, general purpose computing capability which would best satisfy its overall staff requirements within the limits of its resources. This change requires an adjustment by all parties concerned and the transition will take time to accomplish. It will be interesting to see what changes in our working environment and relationships result.

These are several technological changes that may affect scientific computing in the near future. Others should be included, also. The availability of large data files, improvement in man-machine interaction, progress in data representation (including the question of what data should be stored and what should be reconstructed), and others will have a significant impact.

Yet each of these seems likely to have effects which are rather specific to particular fields of research, in the near term, and little influence common to all fields. Color graphics may be an exception.

One other project, which NSF is undertaking, should also be mentioned. That is a cooperative effort among several universities and national laboratories to analyze, certify, and document standard subroutines. The resulting library should give us greater basis for confidence in results and less trauma in carrying out research programs involving computing.

Before closing, I would like to mention the problem of publication. Scientific computing has labored under a handicap because it has not been possible to publish programs. A number of factors are relevant but I am not sure of their causal ordering.

(1) *Attitude*

Programming has been considered necessary but menial even though it has been the principal occupation of many scientists in terms of time and energy.

(2) *Language*

Mathematics has a language which is concise and makes the intellectual content clear. Often, the best description of a FORTRAN program is the program itself which is neither concise nor clear.

(3) *Utility*

A mathematical analysis of a theory or experiment can be universally understood, generalized, and used. A program is typically obscure, specific, and almost universally useless.

The belittling attitude toward programming, which is another carry-over of tradition, has certainly inhibited any attempt to solve the problem but failure to overcome the other hurdles does not encourage a change in attitude. It is a classical example of an impasse, one which natural scientists will not solve within the framework of their own disciplines.

It may be that computer science will make an important contribution here by developing the basis for describing and communicating programs in a really useful and general way. If that results, the benefit to science of improved communication may be great. It is interesting, though, that the path to this accomplishment will have been through byways such as automata theory, formal languages, perhaps set theory, the study of algorithms, and the structure of machines which have almost no discernible relevance to physical science.

We have seen that although progress in large scale scientific computing has been surprisingly rapid, it has also been bound by tradition. We have also seen that there is a long time constant in developing and absorbing technology. Both of these facts tend to become obscured by the activity around us but together they will continue to shape the near term future. At the same time, a new influence is emerging through the development of computer science which may provide the means to relax the constraints of tradition and expand both our conceptual and procedural horizons. The time scale for fundamental change, if it occurs, can be expected to be long but a decade may reveal the strength of the trend. It will be interesting to watch.

Computer architecture and very large problems*

by R. B. LAZARUS

University of California
Los Alamos, New Mexico

For the purposes of this paper, "very large problems" are defined by two properties: (1) There is an effectively unlimited amount of physics and of spatial detail which it would be useful to include if practical; and (2) the calculation must be completed in a limited number of hours. The best example is perhaps the global weather problem. We will obviously never put in all the energy sources or all the topography and we must obviously calculate faster than real time.

Such problems need big machines in the sense of operating as rapidly as possible on large numbers of variables which describe or pertain to a single region of space. It is the aim of this paper to consider questions of computer architecture in this context.

The questions of primary interest are (1) what computer architecture is best for such problems today, and (2) what architecture may be best ten years from now. These questions will not, in fact, be answered. The following comments may, however, indicate why the first question is unanswerable and help others in a search for the answer to the second.

To keep the problem within a reasonable scope, it will be assumed as a boundary condition that the cost per unit of arithmetic of the ideal computer for large problems must not be substantially greater than the cost per unit of arithmetic of, say, an IBM 360/195 or a CDC 7600. This assumption is intended to exclude an architecture which offers a factor of n in speed if the cost is up by a factor of, say, $3n$. This admittedly excludes an interesting area; if, for example, it becomes necessary to control Los Angeles air traffic by computer and the best available computer is too slow by a factor of three, then it would certainly be interesting to know whether the necessary computer could be built for a cost up by a factor of nine. Nonetheless, the restricted question would appear to be of sufficient interest, since many large problems, though important, command only limited funds.

Let us start by considering the role of logic elements. They differ in cost, speed, size, power dissipation, sensitivity, and reliability. In many cases, there are inherent trade-offs among these properties. Let us imagine an omniscient computer designer who has in mind all possible computer architectures—no, make that all possible computer designs—and who has full knowledge of all available logic elements. For all possible designs implemented with all available types of element, he wants to find that combination giving the maximum speed, subject only to the restrictions that speed-to-cost ratio be not much higher than what is now available and that the computer have a mean time to failure of many hours.

He would immediately realize that the problem is not well defined. He would ask what *kind* of calculation he is to maximize the speed of, and be told, "Large problems, as defined above." Noting that such problems have "large numbers of variables," he would fold into his task the consideration of the various kinds of memories available. He would then come up against the following major architectural problem.

Define access time as the time which must elapse between a decision to fetch an operand from a given address and the arrival of that operand at the desired function unit. Then, for all large-memory technologies today, the access time is long compared to the achievable average instruction execution time. (In other words, you can't win by rebuilding the IBM 704 with modern components.) What can be done? There appear to be only two things to do. First, one can try to minimize the number of operand fetches required during the solution of the given problem. Second, one can overlap the fetching of operands, both with each other and with computation.

It is interesting to ask whether these two things are in conflict with each other. For machines with internal

* This work performed under the auspices of the U. S. Atomic Energy Commission.

addressable registers, good coders (and compilers) exploit the first; by clever use of the registers, they try to minimize the number of times a given operand is fetched. For the upcoming vector machines, on the other hand, good coders will try to exploit the second; they will maximize overlap by favoring the vector commands, which stream operands from memory as fast as the memories and memory buses permit.

Is there, then, a conflict or not? There is a conflict only so long as the restriction on gate count (which may come from cost, size, power, or reliability) prevents us from having the best of both worlds. Here is precisely where the question gets so sticky. Are we discussing today's practical problems, or are we looking for fundamentals? This matter will be touched on again below.

In passing, it might be noted that this problem of long access time cannot be solved merely by development of a very fast memory, because transmission times for address and data would still be long. It is no help to postulate a general scaling down of the whole computer, so that all function units can be very close to the memory, because this will scale down the instruction execution time as well. A very restricted solution is suggested in the Appendix, but just for fun, since it certainly cannot be achieved within the next ten years. The long access time should probably be accepted as a fact of life.

At this point, our omniscient designer might note that, while memory access times are inherently long, it is relatively easy to achieve high bandwidth by a variety of methods. He might then pause to consider the methods appropriate to various architectures and note advantages and disadvantages.

The first technique uses a memory word which is two or more machine words in length. In all plausible situations, the number is a power of two and the machine words have consecutive effective addresses. For a given memory technology, this increases *potential* bandwidth as long as the read time increases more slowly than the size of the memory word. Whether or not it increases *realized* bandwidth depends on the mean number of *useful* machine words which are acquired by a fetch of a memory word. The technique is obviously appropriate for instruction fetching and for feeding streams of consecutive operands to a vector machine. With many independent memory boxes (see next paragraph), the technique is also quite powerful for most "naturally occurring" codes, if the boxes have fast registers to hold the last memory word fetched, or in the presence of a "cache" memory as in the IBM 360/85 and 195.

The second technique uses interleaving of many independent memory boxes. We may assume with no loss of generality that the number of boxes is a power of two and that they are interleaved on the low order bits of the memory word addresses. This technique is also obviously appropriate for feeding streams of consecutive operands to a vector machine. Let us examine the behavior of such a memory for a sequence of fetches from random addresses; for simplicity, assume there is only one machine word per memory word. Suppose we have an infinite list of random addresses, a clock, and the logic to pass one address per cycle to the appropriate memory box. If the order in which the fetches get accomplished is irrelevant (and if we have sufficient logic), then our steady state will either be one fetch per clock cycle or else all boxes working to capacity, depending on whether the clock is relatively slow or relatively fast compared to the memory cycle time. In either event, that steady state will constitute maximum available data rate.

If our list has precedence relations (as it certainly must when we mix in stores as well as fetches), or if we do not have facilities to queue up an arbitrarily long list, or if we do not have the logic to scan an arbitrarily long list in one clock cycle, then we will have conflicts and loss of data rate. It is probably not fruitful to derive results for random addresses, since there is a great deal of correlation in real cases.

It is worth noting that additional processors may be able to share a memory "for free" whenever a single processor leaves most boxes idle most of the time. Such idleness could be because of a relatively slow clock cycle or very many boxes, or it could be because of too many precedence relations in the work of a single processor (assuming that multiple processors have no, or at least negligible, precedence relations among each other).

The third technique uses separate memories serving separate processors. This is the technique of the ILLIAC IV, where each memory-processor pair works on a separate portion of the large problem. The potential bandwidth is simply the product of the number of such pairs and the bandwidth of an individual pair. The realized bandwidth is less by the mean fraction of processors doing useful work.

By this time, our designer, being merely omniscient and not divine, has developed a headache. He returns to his consideration of logic elements, having in mind that there are several ways to get whatever bandwidth he needs from memory. He takes a quick look at the possibility of using the best of everything. He imagines a design of eight super-processors with independent instruction streams, where each super-processor consists of 64 slave processors, each combining the arithmetic capabilities of a 7600 with vector streaming capabilities. Before bothering to add up the price, he sees that the mean time between failures will be a few minutes.

Concluding, then, that he must compromise, that he may use only a subset of the techniques which enhance speed for certain kinds of work, he realizes that his problem is still not well defined. He calls in a friend who knows all about programming, numerical analysis, and applied mathematics. He asks for guidance on the fundamental arithmetic and data flow requirements of the large problems under consideration.

Unfortunately, his friend points out two difficulties. First of all, there are trade-offs between the amounts of arithmetic and the amounts of data flow, at both the programming and mathematical levels. Second, the properties of any important computer which may come into existence will themselves influence the development of mathematical techniques.

The second point is brushed aside, and a new problem, this time well defined, is stated as follows. For all possible computer designs, implemented with all available types of element and applied to today's large problems with all well-known mathematical methods, which combination of design, implementation, and calculational method gives the fastest solution, subject again to cost and reliability restrictions?

Perhaps these gentlemen are busy somewhere solving this well defined problem. Unfortunately, they have not been heard from, and the problem is too difficult for ordinary mortals. What should be done?

What actually has been done in recent years is as follows. Various designers have chosen some medium-sized area in the design-implementation space, looked for a local optimum with respect to some set of pieces of problems, and proceeded to create a computer. During the creation time, mathematicians have made some exploration of the calculational method (and programming technique) space by analysis and simulation. After creation (if successful), programmers can be expected to find quite rapidly the actual optimum with respect to programming technique; over the years, the mathematicians will also move us steadily toward the optimum with respect to calculational method. In this way, points are located empirically in the space of interest, namely the design-implementation-method space.

It is time now to look more specifically at the present state of affairs. Consider the approximately 16 year interval between the IBM 701 and the CDC 7600. A typical large problem ran for about ten hours on the 701; a typical large problem runs for about ten hours on the 7600. The ratio of work being done is perhaps 1000; the cost ratio is perhaps five, leaving an improvement factor of 200. Most of this factor, which amounts to 40 percent per year, comes from raw component speed. The rest comes from improved design (architecture, if you wish), including floating point arithmetic, index registers, parallel input-output, sophisticated arithmetic engines, and, finally, parallelism among instruction decoding, addressing, data flow, and arithmetic. *Most* of what came from improved design was again due to improved component technology, being obvious enough in principle but requiring large component counts which, in turn, required cheap, small, reliable components. For the large problems, not very much at all, up through the 7600, came from what we would now like to call really new architectural concepts, like vector arithmetic, multi-processors, or parallel processors.

The reason why architecture has apparently played so small a role, to date, for large problems is precisely that 40 percent per year is very big. If implementation of a substantially new architecture requires a 50 percent increase in cost and a two year delay, one is giving away a factor of three to begin with. Furthermore, if the architecture moves us away from the optimum in method space (i.e., if new methods and/or reprogramming is required), one gives away a further factor; it is generally considered that at least a factor of four must be anticipated in order to motivate a substantial change in method or program for a large problem.

To say this again in other words: As long as 40 percent per year improvement keeps coming along "for free," the *short term* needs of the large problems under consideration here do not motivate a substantial architectural change unless the anticipated speed improvement is substantial. "Substantial" means a factor of four in speed-to-cost over the competition *after* correcting for delivery date at 40 percent per year, compounded.

There are, nonetheless, motivations. The first is the need to shift onto a more rapidly rising curve, if there is one, even if one takes an initial loss to do so. If routine improvements will come at a greater rate with some new architecture than with old architectures, then the short term loss from making the transition will eventually be overbalanced.

The second and more important motivation is the need to explore the design-implementation space in order to get ready for the time, which must come sooner or later, when nothing approaching 40 percent per year will be available in the implementation direction. The trouble is knowing where to explore, and how much to spend on such exploration.

To begin with, it must be noted that the manufacturers will continue to explore, through actual construction, architectures which look profitable. Although the nature of the expanding computer market is such that architectures will not look profitable if they are only good for large problems of the type considered here, some light will nonetheless be shed on our question of the best machine for such problems. It will not be shed optimally, but it will be shed, in some sense, for free.

There is a key difference in the importance of time between the commercial interest and the computer science interest. In the fullness of time, it will make no significant difference to computer science whether something was achieved a year or two earlier or later. But it can make quite a substantial difference to the company which achieves it first.

All in all, this author sees no valid basis for criticizing the nature of the action within the private sector. On the other hand, it will be appropriate from time to time for the Government to underwrite exploration of important areas where the chance for profit is too far off to attract private industry. Some comments may be in order on how such exploration should take place. The particular question of interest to this author is whether the exploration should rely on simulation or on machine construction.

Some years ago it was argued plausibly that the effective speed, for real large problems, of a new architecture could only be determined by implementing that architecture in a real computer as big and fast as anything else available. The argument was that only then would the appropriate people be motivated to optimize methods and programming. This may well have been true some years ago, but it is probably not true now. Two relevant things have increased in the last few years, namely the ability to do simulation at reasonable cost and the number of unemployed scientists.

CONCLUSION

To find the best computer, one must explore the space of computer design, componentry, mathematical method, and, programming technique. For a given computer design (or, at least, architecture), there has been substantial progress from improved componentry. This progress will continue for some time, but not forever. There has also been substantial progress from improved methods and programming techniques. This progress will also continue for some time, quite possibly forever.

On the other hand, one cannot say that, for a given componentry, there has been or is about to be substantial progress from improved architecture. Part of the reason is that the 40 percent per year of progress which has been available without architectural change puts a handicap on any development which requires more time (because it is breaking new ground) between choice of components and delivery. The rest of the reason must be that no new architecture has been invented which offers an order of magnitude improvement.

Some day, the rate of increase of component speeds must become small. At that time, architectural changes which give small improvements in performance will be profitable. In this area, it seems safe to assume that exploration by industry will be adequate.

If, on the other hand, as seems likely, the practical limit on gate count goes up by several orders of magnitude, substantial improvement will be available from architectural changes, and only from architectural changes.

It is on this basis of very large gate count that architectural exploration will be needed, in the opinion of the author. If such exploration is to be carried out be"ore the large gate counts are actually available, it must of course be done by simulation. The right time to begin is probably about ten years before the results are needed. Is that today? Was it, perhaps, a few years ago?

APPENDIX

If a memory technology came about which could give *one* function unit very fast access to all the words in a large memory, if there were an architecture with a crucial need for such very fast access from n function units, if such access to a small (e.g., register) memory were already in hand, and if the memory technology in question were so cheap that one could afford n-fold memory redundancy, then one might consider the following.

Let each function unit have its own memory, and let each memory start out with identical contents. Define two types of storing, A and B. A Type A store is the ordinary type, in which, logically, the old value of some variable is instantly replaced by the new value. A Type B store goes only to "empty" cell (i.e, it replaces dead, or garbage, information) and the information is not going to be refetchable for a long time. Then require all problems to restrict Type A stores to the small memory and make only Type B stores to the large memory.

Many readers will lose interest at this point; there are, nonetheless, many large problems which fit these rules. What is proposed, obviously, is that Type B stores be propagated from memory to memory at leisure. The whole thing would likely be of interest only if this propagation came from a natural diffusion inherent in the physics of the memory. For example, imagine that the binary storage element is a cell in a long filament. The filament is stable with all its cells zero or all its cells one. If any cell is changed, the change propagates along the filament in both directions. Each function unit gets at all the filaments at some convenient station along their length.

Scientific applications of computers in the '70s

by M. STUART LYNN

Rice University
Houston, Texas

It is still fashionable to predict that what happens in the future of computing will be a glorified extension of what has happened in the past. Thus, the last decade has seen an explosion in all uses of computers and therefore, so the fashion would dictate, this explosion must necessarily continue if the faith is not to be shaken.

There is no doubt that exciting new ways of putting computers to work will be found. This writer is in no better position than anyone else to forecast what these will be. It will, however, be the main thesis of this discussion that the explosive rate of growth will *not* continue, particularly in the area of scientific applications, but that the emphasis will be on making known applications work, and work in a more meaningful environment.

GOLDFIELDS

The gold-rush of the sixties is over—this period was characterized by taking a field of man's endeavor, any field, multiplying it by three and attempting to automate it, independent of whether or not such automation could in any way be justified. Support was easily available for anything and, as such practically every field of man's interest was a candidate for computer application. The fact that reality has fallen short of promise has left a skeptical, if not nasty, taste in many mouths. It is probably true, however, that we have been left with a more realistic perspective as to the meaningful uses of computers and, even more importantly, the limitations of such usage.

The gap between promise and reality is, for example, evident in application areas dependent upon modelling or simulation. Ten years ago it was customary to insist that we were on the verge of modelling the world and its component parts to fifteen decimal places; all that was required was another substantial injection of federal or other support to achieve the necessary breakthrough. In application after application a similar pattern was followed: early promise with highly simplified models indicated great things to come, only to be followed in time with the limitations of reality, as the complexities of highly nonlinear or otherwise complex problems and indeterminate data circumscribed the bounds of possibility. The debris left behind in many of the goldfields became depressingly familiar.

Now we no longer attempt fifteen decimal places. We talk instead of provoking lines of thought which might not otherwise have occurred. This is probably a healthy trend, although it is perhaps unfortunate that we are still not learning from the past in some newer application areas.

INDUSTRY TRENDS

In addition to the effects of left-over skepticism, trends in the computer industry will also mitigate against a continuation of the applications explosion. It is certainly highly probable that the current computer overcapacity in this country, when combined with the decreasing cost/performance curve of computer systems, will cause considerable pressure on the industry to underwrite new applications as a means of supporting continued expansion of the industry. However:

- It is likely that the most acceptable applications will be those whose economic pay-off is readily justifiable in the short-term, in view of past management experiences and of the current economic climate. This would tend not to favor scientific applications.

- The effects of unbundling have already moved the onus of applications research and development more than ever on to the end-user. This will continue. Whereas this has little impact on the traditional user of computers, it reduces the prospects of important uses of computers materializing among new user classes.
- The increasing pressure for rapid economic justification lessens the changes of serendipity yielding new applications. The principal opportunities for serendipity will continue to move back to the universities; this is probably not a bad thing, except that universities are not always in the best position to judge user needs over the long-term.

WHAT WILL HAPPEN?

This writer suggests that the following are likely to be characteristic of scientific applications of the next decade:

(1) The areas of application familiar today are unlikely to change substantially. The relative emphasis between areas is, however, likely to change as a reflection of society's changing priorities, as has already been observable over the past three years. Thus, for example, as a reflection of NASA's changing priorities, the use of computers in support of this nation's activities in space will shift toward the many possibilities involved in applications relating to the Earth Resources Program, a shift that has already begun to take place. Again by way of example, the ability to simulate ever more complex molecules and chemical reactions will have important applications in the petroleum, chemical and pharmaceutical industries.

(2) Basis technological developments in support of applications will not mature at anywhere near the pace of the sixties. The emphasis will shift toward making fuller use of existing technologies and in making them support each other. Thus, for example, the concern of the sixties for speed and compactness in algorithmic processes will have diminished importance in the complex environments of present and future operating systems and as the cost of computing continues to decrease. On the other hand, the unnatural boundaries between such areas as numerical analysis, mathematical programming, computational statistics, signal analysis, pattern recognition and control theory will continue to become less well-defined.

(3) There will be more emphasis on total systems implementation of known applications so that they are economically (or otherwise) useful to the end-user. This will include careful exploration of computing environments in order to minimize overall costs and to maximize, *inter alia,* user productivity and convenience. In this context there will continue to be an increasing emphasis upon reliable systems. Substantial investment in research and development of scientific computer applications will be closely geared to their proven practicality and utility in society as a whole.

(4) Basic methodologies already well understood in established application areas will be brought to bear upon newer areas. Examples of this are to be found in the electric power industry and in water estuarial modelling where only too often the simulation techniques currently used are primitive by comparison with what has been developed in other application areas.

(5) The computer will continue to be brought to bear, only with ever increasing emphasis, in making the scientific research process itself more efficient. Advances in the technology of interfacing with computers will become sufficiently reliable and efficient that the use of such technology will become widespread (many scientists use computers today no differently than they did ten years ago, in spite of the changes that have taken place). The ability to interact efficiently with large data bases, the growth of extensible languages, the development of special processors configurable by the user to meet his particular requirements, are examples of technologies which will have an impact upon ways in which scientists use computers.

This appraisal is not intended to be pessimistic. Within the above confines, the opportunities for using computers in exciting ways should be as great, if not greater, than ever before. The challenge, however, is one of turning more of the promises of the past decade into accepted reality.

The functions and elements of a training system

by B. A. JONES

Bankers Trust Company
New York, New York

"From a systems point of view, the design of an operation which can successfully carry out the training function becomes a problem of creating a system to accomplish a given end result or objective. In the case of training, this end result is to effect a group of planned and predetermined behavior changes in the organization."[1]

The following are five basic functions that I feel are essential for an effective training system that will allow you to effect that group of predetermined behavior changes:

I. Training Analysis
II. Educational Consulting
III. Training System Support
IV. Public Relations
V. Management

The elements of each of these functions become the procedures necessary for the successful operation of the entire training system. To better understand these five basic functions it is necessary to look at their elements.

The training analysis function (Figure 1) and its elements is the critical activity in determining training requirements.

We begin the analysis by gathering valid data. That is, in terms of performance, what is it you want the trainee to do that he is not now doing, or what is it you want him to do better? Determine whether these are basic skills and knowledge he should possess or whether they are supplemental skills and knowledge that will upgrade his proficiency. This may determine the priority of training.

The second element in the training analysis function is to address performance standards by conducting a deficiency analysis. At this point the following determinations *must* be made:

1. Is this a training problem?
2. Is this a feedback problem?
3. Is this an attitude problem?
4. Is this due to a lack of resources?
5. Is this due to a lack of reinforcement?
6. Is this a procedures problem?

When this element of the training analysis has been completed, the suggested solution to the initial request or desired end result may be made. If the suggested solution is to develop a training curriculum, the third element in the training analysis, that of stating the terminal behaviors, begins.

Once this has been accomplished, the design of the training curriculum proceeds. It is at this point that the cost of training, the length of training, the media, the method of evaluation, may be determined.

The implementation and evaluation of the training curriculum *may* end the analysis function. If the desired end result was not obtained, it may be due to a bad analysis or a poor design. Whichever is the case, a reiteration through the process is essential. If the desired end result has been met, the final element is to maintain the material. That is, if it is a one time only curriculum, it is placed in a history file. If it is an ongoing curriculum, it is kept up-to-date.

When the suggested solution, the second element, is *not* the design of a training curriculum, the Educational Consulting function becomes crucial. Figure 2 illustrates the elements of this function. It differs from the training analysis in that the training person acts only as an advisor, i.e., he can suggest solutions but is not accountable for implementation or evaluation. Figure 3 lists some of the skills and knowledge necessary for successfully performing the consultation function. The questions asked during the follow-up emphasize the advisory capacity of the consultant.

The third function of the training system, Training System Support, is basically an administrative and maintenance activity. However, the support is essential for a training division if it is to effectively serve and

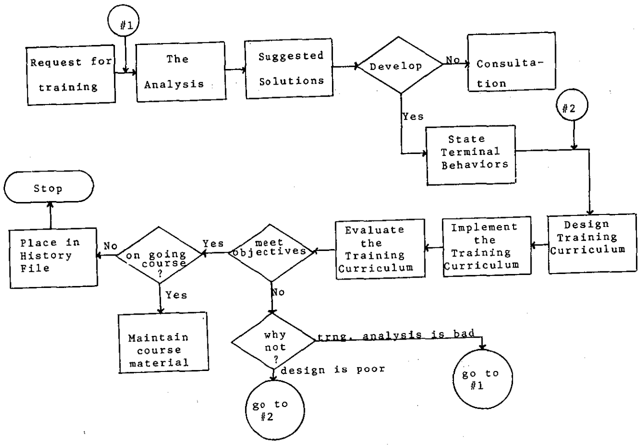


Figure 1—Training analysis function

assist its users. The elements of this function are exhaustless, but three of these elements are vital.

Record keeping provides an historical reference for each student and aids in defining what courses he must yet complete in terms of his career development. Record keeping is of great importance also in prepar-

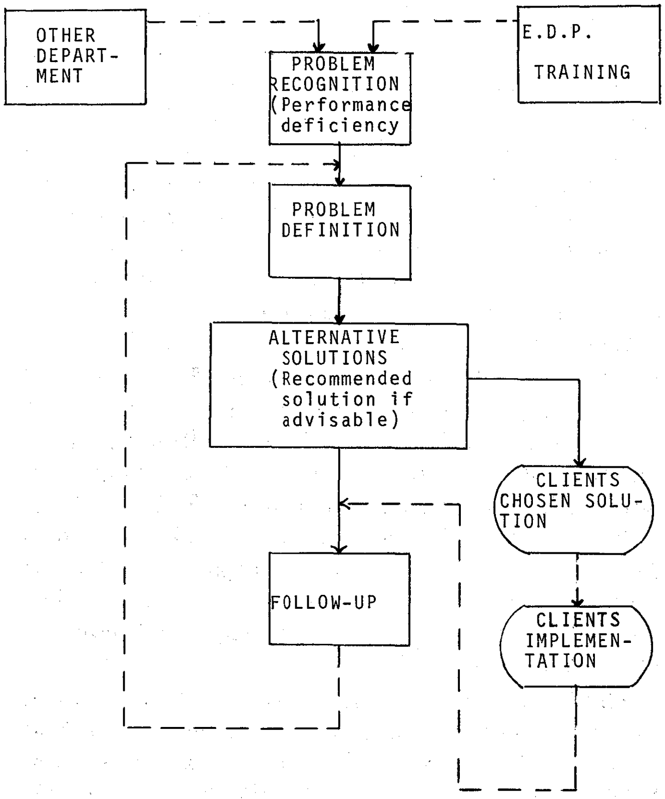


Figure 2—Consultation function

Sell
Interview
Problem Solving
Advise
Help
Expertise in training
Communication
Analysis
Suggestions
Persuade
Confident
Confidentiality
Patience
Sensitivity

Follow-up
—did you decide?
—what did you decide?
—were you successful?
—may I help you?

Figure 3—Consultation skills and knowledges

Name______________________ Course Date________________

Title______________________ Phone________________

Department______________ Uptown
Downtown

Division______________

Supervisor's signature______________________________

Current Responsibilities:______________________________

____

____

____

The following information is helpful to the course instructors and they would appreciate your cooperation in supplying it.

EXPERIENCE

Number of years/months in any Data Processing Function:

Programmer______________

Systems Analyst____________

Other (please state)______________________________

EDUCATION

Check highest Degree and indicate major:

High School______________ Undergraduate____________

Graduate________________ Ph.D.________________

Major__________________

Figure 4—Systems analysis and design—Enrollment form

ing a cost per student figure, developing and justifying a yearly budget, and forecasting resources. Figure 4 illustrates such a form.

Equipment maintenance is necessary for an effective and efficient training facility. It doesn't matter if the facility has only a one person training staff, a conference room, and an overhead projector, or is a large scale multi-media facility with a full-time training staff. Calvin R. Gould, put it this way:

> A national meeting of professional communicators was ending its second day with a banquet that featured a learned and distinguished individual. Before the banquet, the speaker's assistant set up a pair of 35mm projectors and a screen. This person would be the projectionist and would work from a cued copy of the speech. While this practice requires that the speaker read his material, it is more effective than asking for each slide, and if well rehearsed, can be professionally done. The audience quite properly expected a worthwhile message from the speaker, a vice president of a large manufacturing company. They weren't to be denied. His initial remarks, charging the audience with their responsibility to world communications, proved he was going to be a dynamic speaker. As the first slide was projected the house lights went off right on cue. Everything was working like clockwork, even the pause that followed the first slide seemed effective and timely . . . except it seemed to be longer than necessary . . . really too long. Finally the audience, the projectionist, and the speaker all seemed to realize simultaneously that the speaker's lectern light went out with the house lights. The fifteen minutes interruption required to recover from this unfortunate situation was enough to seriously damage the communications effectiveness of this presentation. I was in the audience, and I have long forgotten the message.[2]

A schedule for the maintenance of equipment must be established, fully documented and enforced. It should include timetables for the changing of bulbs, the cleaning of equipment, periodic maintenance checks, etc.

Inventory control is important, since in the majority of installations the training division has equipment that can be borrowed by other departments. The unavailability of equipment can often lead to the cancellation of a class. The availability of existing equipment should be documented by an inventory and status report form. Figure 5 illustrates the form used at Bankers Trust.

DESCRIPTION	QUANTITY	CONDITION	AVAILABILITY
SLIDE PROJECTORS	3	GOOD	YES
16MM PROJECTORS	1	GOOD	NO
16MM CAMERA (BOLEX)	1	GOOD	NO
VIDEOCORDER (1″ & 2″)	3	GOOD	½″ only
VIDEOCAMERA	1	SATISFACTORY	YES
T.V. MONITORS	5	GOOD	3 only
MARK IV	1	GOOD	NO
CASSETTE PLAYER	2	GOOD	YES
HEADPHONE SETS	4	GOOD	NO
3M FOIL MAKER	1	GOOD	YES
3M OVERHEAD PROJECTOR	1	SATISFACTORY	YES
PORTABLE SCREENS	1	GOOD	YES
6 FOOT MIC	1	GOOD	NO
NECK MIC	1	GOOD	NO
MARK V	1	GOOD	NO
35MM CAMERA	1	REPAIR	NO
MP-3 CAMERA	1	GOOD	NO

Figure 5—Inventory and status report

The Public Relations function has two distinct elements. Within the organization, communication is with users of data processing training. The promotion of in-house courses and the *selling* of services are necessary in order to establish and maintain credibility. The external element is, on the one hand, *to let the rest of*

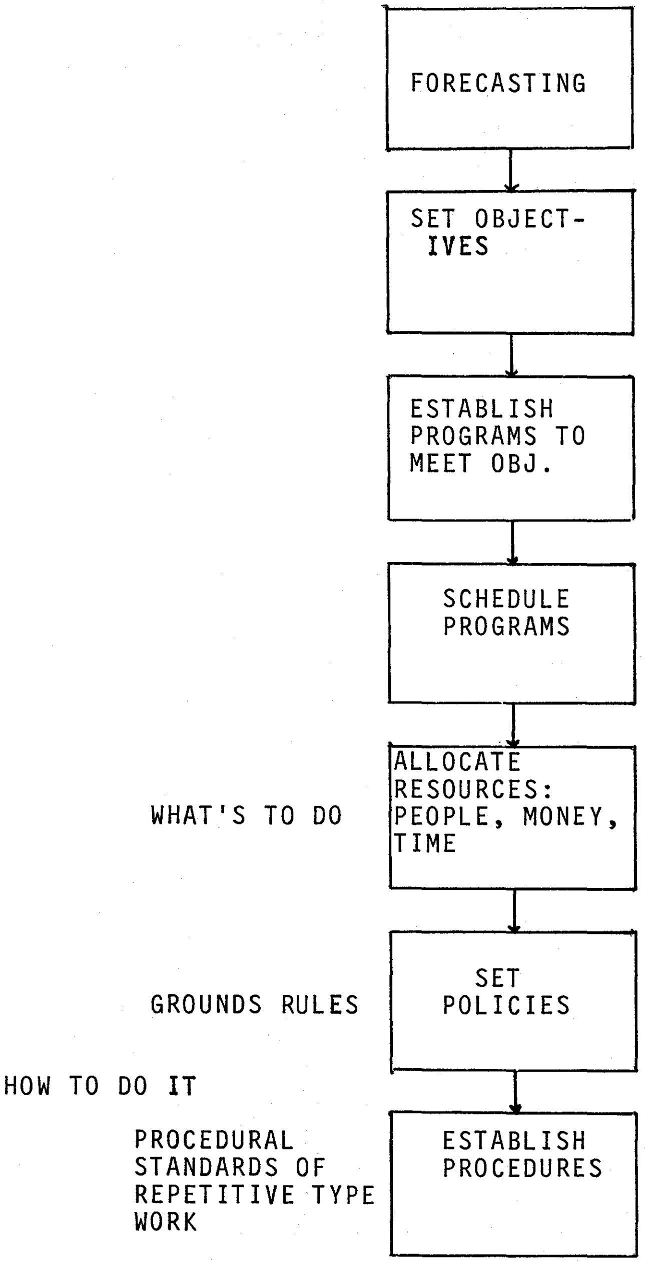

FORECASTING
SET OBJECTIVES
ESTABLISH PROGRAMS TO MEET OBJ.
SCHEDULE PROGRAMS
ALLOCATE RESOURCES: PEOPLE, MONEY, TIME
WHAT'S TO DO
SET POLICIES
GROUNDS RULES
HOW TO DO IT
ESTABLISH PROCEDURES
PROCEDURAL STANDARDS OF REPETITIVE TYPE WORK

Figure 6—Management function planning sub-function

the world know what your organization is doing, and, on the other, to make sure that you yourself are not re-inventing the wheel.

The fifth function, that activity which permits the combination of time, people, and money to efficiently achieve the desired end result, is the Management function. It has four major elements: Planning, Organizing, Leading, and Controlling. Within each element are specific activities.

Since most data processing training staffs are not expert in management philosophy or management development, yet find it necessary to *manage* a training system, I offer the following descriptions of the four elements in the management function from Louis A. Allen, president of Louis A. Allen Associates, Inc.

I. Planning

"Planning is the work a manager does to master the future. Planning requires that a manager think before acting. In the process of doing this, he will map out beforehand what he expects to accomplish, how he can best do it, and the limits of cost and other factors that are significant. The principles of planning stress the importance of current decisions in limiting our potential range of action in the future.

They point out that the further we project a plan into the future, the more unstable and undependable it becomes. A key principle emphasizes the tendency of people to resist changes. Resistance to change can be overcome primarily by providing for participation and communication in the planning process. Plans are considered in terms of the specific activities of forecasting, establishing objectives, establishing programs, scheduling programs, allocating resources, setting policies, and establishing procedures."[3]

Figure 6 illustrates how our training staff at Bankers Trust interprets this element.

II. Organizing

"Organizing is the work a manager does to arrange and relate the work to be performed so that it can be carried out most effectively by people. An organization is not people but the mechanism arranged to enable people to work most effectively together. An organization chart is helpful but the chart itself is not the work of

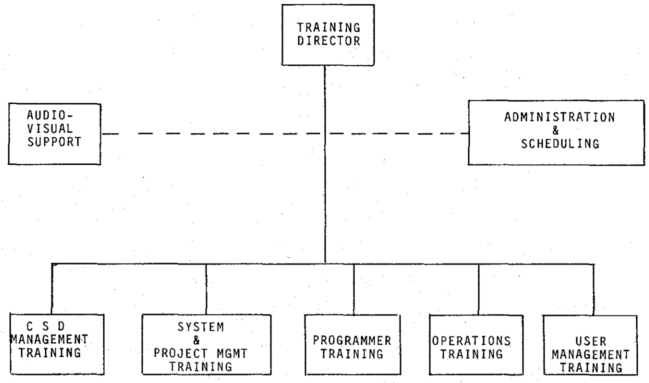


Figure 7—E.D.P. training division

organizing, for this is how one insures that the most work is accomplished by the fewest people at the least cost."[4]

Figure 7 illustrates the organization of our training division.

III. Leading

"Leading is the work a manager performs to get people to take required action. Leading is the energizing function that makes planning, control, and organizing possible. In studying what it takes to lead, we find that there is no one best management personality. The proper blend of authoritarian and democratic leadership knowingly applied is best. The specific activities within this function are decision making, communications motivating people, selecting people, and the development of people. We are at the present time within our element of control. A lot of work has yet to be accomplished so that these two elements of management will relate directly to our responsibilities."[5]

IV. Controlling

"Controlling is the work a manager does to assess and regulate work in progress and completed. The control is the manager's means of checking up.

While control by personal observation is often necessary, the most desirable method is control by exception. Here the principle of Least Cause and the principle of Point of Control are important to ensure that the greatest amount of control is applied where it will do the most good. The significant activities inherent in good control are establishing performance standards, performance measuring, performance evaluating, and performance correcting."[6]

It is apparent that, due to the sheer length of the process, a training staff of more than one is desirable. However, given that the training needs are clearly defined and not too abundant, and that time and money are available, such a process can be implemented effectively by a minimal staff. To a great extent, this is due to the increasing availability of vendor-supplied training materials, and also to new, multi-media techniques for its presentation.

To properly select media that will meet training objectives, one must understand the advantages and disadvantages of each. The most commonly used (and misused) media are the spoken and written word. Several studies have indicated that we retain only 20 percent of what we hear. It has also been cited that the average student possesses a reading level of eighth grade or lower. Therefore, it is necessary for the trainer to know his students' abilities and to adjust his communication methods accordingly.

The studies on learning further suggests that we retain 80 percent of what we see and as much as 95 percent of what we both hear and see. Thus, it appears that a lecture supported with various media would allow us to satisfy our objectives in the most efficient and effective manner.

It has been my experience, being fortunate enough to have access to a very sophisticated multi-media training facility, that the integration of various media into our training system has accomplished several things:

1. This integration makes use of two or more senses and
2. More easily establishes a relationship between the unknown and the known.
3. Often, visuals can save teaching time because they add to the student's ability to arouse and retain interest in the subject.
4. Visuals tend to create a sense of involvement and participation.
5. When color is used in visuals, the retention rate is vastly increased.

There are certain obvious drawbacks when a training facility is heavily involved with the use of audio/visuals. I have stated earlier that the record keeping function is essential in the smooth and effective operation of a training facility; I would like to emphasize this importance when using audio/visual media. The hardware required for the use of audio/visuals must be maintained as "Operational" at all times. It is *most desirable* to have backup for each piece of equipment you are using; the consequences are self-evident.

Staff resources must often be increased to support a multi-media facility, e.g., for maintenance and administration. However, once standards and procedures are developed for maintaining and scheduling equipment, the advantages one can attain from the use of multimedia are considerable.

In summary, when selecting media for communicating to the learner, consider the following procedures:

1. Determine your training objectives *first*, then determine the means whereby you can convey them to the student.
2. Consider the physical constraints; how will they affect the media you have selected? How many students will you have at one time? Where will they be trained?

3. Determine as best you can the collective backgrounds and abilities of your students, and which media will have the greatest appeal and effectiveness for them.
4. Examine carefully the economics of the ideal system, i.e., look carefully at the cost of alternatives.

SUMMARY

A Training System (the process) is a methodology with established functions, procedures, and activities one can follow to effectively and efficiently achieve a desired result.

A properly evaluated multi-media system, when well maintained, will improve the learner's ability to comprehend and retain the training supplied. This will allow the training division to accomplish its objective, that of giving the user of its services what he wants, when and where he wants it.

REFERENCES

1 M W WARREN
Training for results
Reading Massachusetts Addison-Wesley 1969 p 15

2 C R GOULD
Anatomy of a presentation
Audio-Visual Communications V June 1971 p 20

3 L A ALLEN
The management profession
New York McGraw-Hill 1964 p 109

4 Ibid p 173

5 Ibid p 246

6 Ibid p 324

BIBLIOGRAPHY

L A ALLEN
The management profession
New York 1964

M M BROADWELL
The supervisor as an instructor
Massachusetts 1970

L BENNIGSON
The strategy of running temporary projects
Innovation Number twenty-four September 1971 pp 32-40

HARTMAN MATTHES PROEME
Management information systems handbook
New York 1968

O GLASER
The management of training
Massachusetts 1970

M W WARREN
Training for results
Massachusetts 1969

Planning data processing education to meet job requirements

by JAMES O. HAMMOND

IBM Corporation
White Plains, New York

The greatest single expenditure that a data processing organization has today is for its human resource. This has been dramatically illustrated in several well-known studies. The cost of human resources continues to rise in data processing. Computers and systems are becoming more and more complex. It has become extremely difficult for programmers and analysts to know all the essential things about a system. This is especially true with the large, widespread teleprocessing networks. This complexity of sytems is forcing more and more specialization in computer technology. At the same time, small computer staffs find that although a few staff members may know all that is required to support a system, it is frequently difficult to identify job responsibilities in such a way that maximum efficiency can be attained for the benefit of the business.

Most data processing organizations have major effects on their parent company's ability to meet the needs of its business. It becomes of vital importance, therefore, that the data processing or information systems department, as they are frequently called, be managed as efficiently as possible providing the business support to its parent company in the most optimum fashion. Obviously, the management of any data processing department is an important key factor in its success. In this day and age of national economic lull and austerity moves in business to get more for the dollar spent, it has become even more important that management exercise its very best judgment at all times to run the data processing department more as a business and not just as a support function. In other words, have as valid justification that the money spent is worth what is being gained from the output.

There is another extremely important factor to consider in maintaining a high efficiency level in a data processing organization. This factor is the utilization of the staff employees or the bulk of the human resource. If the data processing staff is selected and trained to meet the requirements and needs of the business of the parent company, then a data processing organization has gone a long way toward solving efficiency problems not to mention personnel problems and potential large cost savings that can be gained.

The staff human resource portion of a data processing organization is the one addressed in this paper. More specifically, ways of defining and structuring data processing skills and job responsibilities are explained. Approaches are included on ways of defining an education program to provide the knowledge and skills that may not currently exist in the skills mix of the data processing staff.

Keeping in mind that all the other key factors of operating an efficient, productive data processing organization have not been forgotten, concentrate for a moment on how we might go about putting some form and rationale to human resource selection and training.

One of the most effective ways of relating data processing skills requirements, and in turn DP education, to a data processing organization is to start with the process being used to develop and implement data processing products. Relating skill requirements to the process is especially effective because the process is normally a well established one with which most company management are familiar both inside and outside the DP organization. Since the DP process is the mainstream around which the majority of all other DP activities in a department revolve, it is most likely a well defined and valid process. If the DP department is just being organized or for some other reason the process to be used is not well defined, then first things should be first—the process should be clearly resolved and defined.

The major steps of the DP process will vary from one DP organization to another and, of course, from

Step	Example 1	Example 2	Example 3	Example 4	Example 5
I	Study	Concept	Evaluate	Determine Function	Application Analysis
II	Design	Definition	Concept	Develop System	Design Synthesis
III	Develop	Develop	Definition	Data Collection	Feasibility Validation
IV	Install	Acquisition	Develop	Consider Alternatives	System Design
V		Operational	Operation	Select Solution	Development & Test
VI				Formulate System	Installation
VII				Review	Operation
VIII				Test	Maintenance
IX				Install	
X				Measure	

Figure 1—DP process steps

one company to another. This is expected and even desired if the organization is being primarily tailored to meet the needs of the parent company's business.

Figure 1 shows a selection of DP processes chosen from the DP organization of several companies in the United States. It can be seen, as was just mentioned, that the steps in the processes do vary. It is interesting, however, to notice that in spite of variations in the steps of the process and that some of the steps are called by different names with essentially the same meaning, that there is a thread of commonality through all the processes. There are some process steps that are common to all data processing organizations regardless of size, complexity, systems types, or structure. Three of these common steps are design, install, and maintain.

It is important that the major steps of the process be clearly defined in any given DP organization where the human resources are being studied. Once this is done a framework is established around which the human resource requirements can be structured. The process shown in Figure 2 has been selected for the purpose of illustrating how we can go about relating the human resource requirements to a DP process.

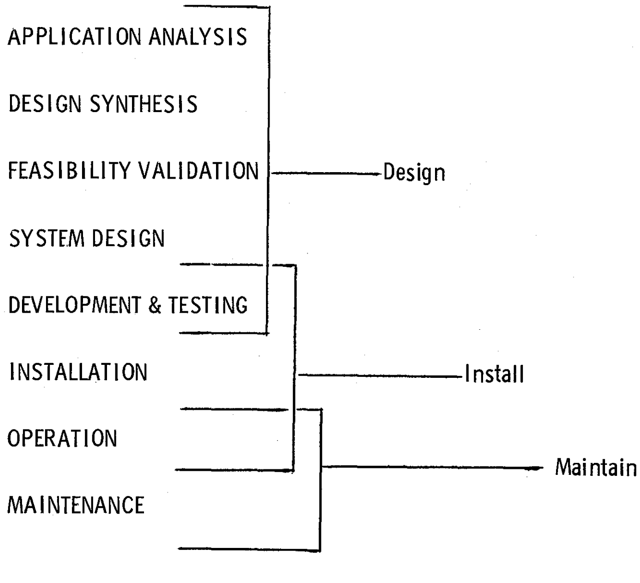


Figure 2—The process

Considering the common thread through the processes again—design, install, and maintain—these general terms can now be expanded into the selected process.

Before we go further in skills requirements structuring, we must make sure that each step of the process is defined well enough so that there is no miscommunication. I have attached the following definitions to each step of this process. Keep in mind that there is no hing fixed about the process and the associated definitions that are being used in this example. Although, according to surveys made, this is one of the more common processes, the steps and definitions may be different for various DP organizations.

So that we may continue our analysis, here are the definitions that will be used.

Process Steps

I. Applications Analysis
II. Design Synthesis
III. Feasibility Validation
IV. System Design
V. Development and Testing
VI. Installation
VII. Operation
VIII. Maintenance

Definitions

I. Applications Analysis

This first step includes defining the problem, analyzing and documenting the existing system, describing the processing, defining output and performance requirements, estimating the value of what is required, and developing and presenting the initial specifications to management for their review and concurrence.

II. Design Synthesis

This step includes establishing project planning and control, checking and refining the systems requirements and initial specifications, developing preliminary design alternatives and selecting a solution that is most economically feasible and best meets the requirements specified (with supporting analysis from Feasibility Validation). The system solution is then proposed to management where a decision must be made whether to proceed into System Design or consider alternative solutions.

III. Feasibility Validation

This includes the processing testing the performance, technical feasibility and validity of design alternatives or existing solutions by creating prototypes, models or through simulation. Major cost and performance improvements can be made possible by the application of standard solutions and generalized program designs already available. Adaptations of existing systems or generalized programs may be entirely adequate solutions; therefore, no further design work would be required.

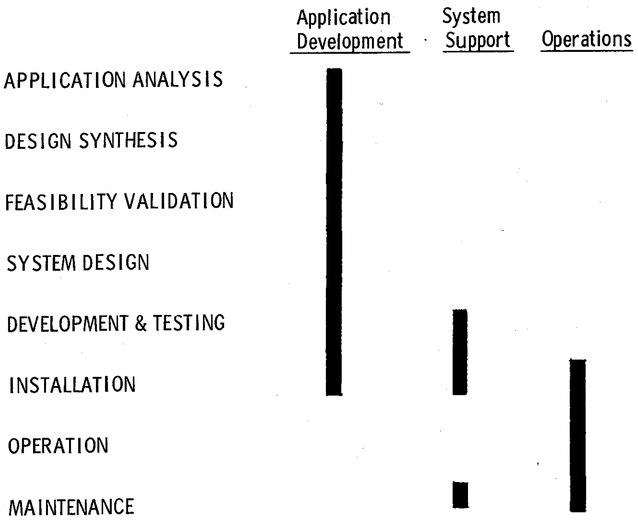


Figure 3—The process expanded

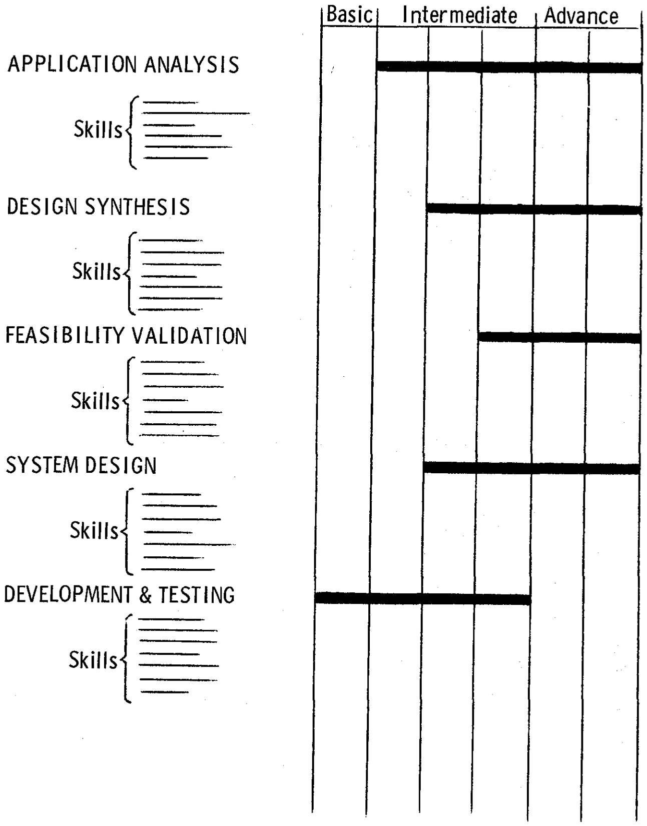


Figure 4—Application development

IV. System Design

This step includes planning and designing the details of the systems solution using the output from Design Synthesis and the feasibility model, selecting hardware and software, developing the new procedures and test plan, and identifying each program.

V. Development and Testing

Here the process is completed and proved by translating the system and program specifications into machine-readable code, testing and debugging this code and documenting each program module.

VI. Installation

This step includes converting data files, transactions, software and hardware used by the existing system to meet the requirements of the new system.

TITLE:
SENIOR ASSOCIATE PROGRAMMER, DEVELOPMENT.

GENERAL CONCEPT:
AN ACTIVE MEMBER OF AN INTERNAL DATA PROCESSING SYSTEM APPLICATION TEAM, RESPONSIBLE FOR DEFINING PROBLEMS, PROPOSING SOLUTIONS, DEVELOPING AND TESTING THOSE SOLUTIONS.
HE IS QUALIFIED TO SPECIALIZE IN THE AREA OF PROGRAM MODULE DEVELOPMENT.

SPECIFIC RESPONSIBILITIES:
(1) THE PRECISE DEFINITION OF PROGRAM MODULES (INPUT, TRANSFORMATION, AND OUTPUT) WHICH WILL IMPLEMENT THE SYSTEM DESCRIBED IN SYSTEM DESIGN.
(2) THE DOCUMENTATION, FLOWCHARTING AND CODING OF THE MODULES SPECIFIED.

COMMUNICATES WITH:
TESTING SPECIALIST FOR RESULTS OF MODULE AND SYSTEM TESTS.
SYSTEM DESIGN SPECIALIST TO REQUEST MODIFICATION OF SYSTEM SPECIFICATIONS, IF THEY ARE FOUND UNCLEAR.
MANAGER TO REPORT PROGRESS AND RECEIVE ASSIGNMENTS.

APPROPRIATE TOOLS, TECHNIQUES AND SKILLS:
ADVANCED PL/1
MODULAR PROGRAM DESIGN
PROGRAMMING TECHNIQUES
OPERATING SYSTEMS INTERFACE

PROMOTION POSSIBILITIES:
STAFF PROGRAMMER, DEVELOPMENT AND TESTING
SENIOR ASSOCIATE PROGRAMMER, SYSTEM DESIGN

Figure 5—Job description

VII. Operation

This step includes running the designed and tested system (or application) on the computer.

VIII. Maintenance

This includes the maintenance of the software systems programs and applications programs, e.g., changes, extensions, and optimization.

There are specific skills, tools and techniques that must be known and applied to accomplish each of the steps in this process. If the organization is a large one, a staff member might specialize in one or two of these steps and be required to master a smaller number of tools, but be able to apply these with considerable expertise. A small DP organization would not have a requirement for such extensive expertise; the staff member may, therefore, be skilled enough to perform several of the steps in the process to a much less level of complexity.

At this point the DP organization functions and job responsibilities can be related to the process that has been described. The organization functions can be identified as being three main segments of activity:

1. Application Development
2. System Support
3. Operations

Figure 3 shows how these organizational functions can be related to the process. Notice the overlap among the functions. For example, the system support function overlaps the application development function in the development and test step and installation step of the process. In large installations where a high degree of specialization may exist, these overlaps indicate the points where responsibilities are passed from one person or group to another. It is extremely crucial that these key points of communication are identified and the responsible staff members are made aware that this "passing of the baton" so to speak, is an important part of their responsibility. It is equally important that they be adequately trained to carry out these tasks with great efficiency.

Now take another cut at the same process from the individual staff member point of view. For the purpose of analyzing how this might be done, consider the applications development function alone for a minute. The staff working within this function of a DP organization

Responsibilities/Skills
(Be Able To:)

Application Analysis

Analyze existing system/applications
Document system
Define system requirements
Development and present initial specifications

Design Synthesis

Refine initial specification
Develop conceptual solutions—design alternatives
Identify and select known/existing solutions
Specify details of each design alternative
Estimate development, conversion, installation and operating costs
Test and evaluate alternatives and select best solution
Present proposed system solution

Feasibility Validation

Create and test prototype systems
Construct and analyze or simulate models
Compare prototype or model to required systems performance

Systems Design

Development systems design work plan
Document system in detail
Design forms and reports
Finalize record and file requirements
Define data editing and validation rules
Select final hardware configuration and software
Document controls, audit trails, back-up procedure
Write new and revise old procedures
Plan system test and prepare test data
Prepare computer operations work flow
Document computer flow and identify each program

Development and Testing

Define program modules
Flowchart, code and document program modules
Design and implement program module tests
Implement system tests specification in system design

Technical Strength Definitions

0 has no knowledge
1 understands capability and applicability
2 can use technique effectively
3 can maintain, modify and create new features
4 can synthesize and evaluate to reduce or eliminate analysis, machine runs, development or tests based on his wide knowledge of alternatives and depth of experience.

Figure 6—Programmer/analyst in application development environment

is made up for the most part of one of three types of personnel: analysts, programmers, and managers. The analysts are known as systems, application or program in most cases. The analysts' job responsibilit'es can range across the analysis, synthesis, validation, and design of DP systems and applications. Programmers are generally of two primary types, applications and systems. Applications programmers program and test primarily business applications whereas system programmers program primarily control and system applications. There are a wide range of job responsibilities performed by programmers at all levels of complexity. The managers referred to here are technical managers. It is important to identify the job responsibilities of higher management along with all the other members of the organization but higher managements' job responsibilities are broader, less technical, and more administrative. For this reason they have been excluded in the sample being used in this paper. The technical manager is one who has direct supervision over a small staff of programmers or analysts and is able to understand technical detail, at least within his assigned responsibilit'es.

There are numerous titles for programmers and

RESPONSIBILITY (SKILLS) (BE ABLE TO:)	BASIC	INTERMEDIATE			ADVANCED	
APPLICATION ANALYSIS						
ANALYZE EXISTING SYSTEM/APPLICATIONS						
DOCUMENT SYSTEM						
DEFINE SYSTEM REQUIREMENTS						
DEVELOP & PRESENT INITIAL SPECIFICATIONS						
DESIGN SYNTHESIS						
REFINE INITIAL SPECIFICATION						
DEVELOP CONCEPTUAL SOLUTIONS—DESIGN ALTERNATIVES						
IDENTIFY AND SELECT KNOWN/EXISTING SOLUTIONS						
SPECIFY DETAILS OF EACH DESIGN ALTERNATIVE						
ESTIMATE DEVELOPMENT, CONVERSION, INSTALLATION AND OPERATION COSTS						
TEST AND EVALUATE ALTERNATIVES AND SELECT BEST SOLUTION						
PRESENT PROPOSED SYSTEM SOLUTION						
FEASIBILITY VALIDATION						
CREATE AND TEST PROTOTYPE SYSTEMS						
CONSTRUCT AND ANALYZE OR SIMULATE MODELS						
COMPARE PROTOTYPE OR MODEL TO REQUIRED SYSTEMS PERFORMANCE						
SYSTEM DESIGN						
DEVELOPMENT SYSTEM DESIGN WORK PLAN	0	1	2	3	4	4
DOCUMENT SYSTEM IN DETAIL	0	2	3	4	4	4
DESIGN FORMS AND REPORTS	1	2	3	4	4	4
FINALIZE RECORD AMD REQUIREMENTS	1	2	3	4	4	4
DEFINE DATA EDITING AND VALIDATION RULES	0	2	3	3	4	4
SELECT FINAL HARDWARE CONFIGURATION AND SOFTWARE	0	2	3	3	4	4
DOCUMENT CONTROLS, AUDIT TRAILS BACK-UP PROCEDURE	0	1	2	3	4	4
WRITE NEW AND REVISE OLD PROCEDURES	1	2	3	4	4	4
PLAN SYSTEM TEST AND PREPARE TEST DATA	0	2	2	3	4	4
PREPARE COMPUTER OPERATIONS WORK FLOW	1	2	3	3	4	4
DOCUMENT COMPUTER FLOW AND IDENTIFY EACH PROGRAM	1	2	3	4	4	4
DEVELOPMENT AND TESTING						
DEFINE PROGRAM MODULES						
FLOWCHART, CODE AND DOCUMENT PROGRAM MODULES						
DESIGN AND IMPLEMENT PROGRAM MODULE TESTS						
IMPLEMENT SYSTEM TESTS SPECIFICATION IN SYSTEM DESIGN						

Figure 7—Programmer/analyst in application development environment responsibility (skills) matrix

analysts. It is best, of course, to define job levels using the titles of a given DP organization. In Figure 4, the levels—basic, intermediate, and advanced,—have been used to relate job levels to the process. The bar graphs under these levels give an indication of the approximate average level required to perform the job responsibilities in the corresponding process steps. For example, most senior staff personnel have had experience and/or training in development and testing before advancing to more senior positions in either systems design, feasibility validation, or design synthesis. This type of structuring forms a rather neat array of potential career paths.

Job levels, titles, and major responsibilities are traditionally structured into a job description. Job descriptions can range in form from the one sheet summary shown in Figure 5 to very detailed descriptions. In all cases, it is worthwhile specifying the

	Basic	Intermediate			Advanced	
System Design						
Develop system design work plan	0	1	2	3	4	4
Document system in detail	0	2	3	4	4	4
Design forms and reports	1	2	3	4	4	4
Finalize record and file requirements	1	2	3	4	4	4
Define data editing and validation rules	0	2	3	3	4	4
Select final hardware configuration and software	0	2	3	3	4	4
Document controls, audit trails, back-up procedure	0	1	2	3	4	4
Write new and revise old procedures	1	2	3	4	4	4
Plan system test and prepare test data	0	2	2	3	4	4
Prepare computer operations work flow	1	2	3	3	4	4
Document computer flow and identify each program	1	2	3	4	4	4

Technical Strength Definitions

0 has no knowledge
1 understands capability and applicability
2 can use technique effectively
3 can maintain, modify and create new features
4 can synthesize and evaluate to reduce or eliminate analysis, machine runs, development or tests based on his wide knowledge of alternatives and depth of experience.

Figure 8—Programmer/analysts—Responsibility matrix expanded

knowledge and skills required as well as promotion possibilities. This information is valuable to management both in utilizing the employees best talents and planning for his career advancement.

A further expansion is necessary of the knowledge and skills required to do the jobs within each step of the process. If some type of job analysis has not been performed on the data processing organization in question, then now is the time. What the job involves, the complexity of the technical level, the associated responsibilities, and, last but not least, the knowledge and skill level required to do the job all must be defined to fulfill the requirements specifed in the steps of the process. Much of this information can be structured on the job description summary sheet that was just described.

Job analysis should be done by professionals who have had experience in making such analysis. This type of study is usually performed for a fee by an outside consultant. If the consultant is carefully chosen the results will be well worth the money but be sure that he has extensive data processing expertise on his staff to participate in the analysis. Also make sure that his analysis is going to be provided to you in a form that fits your needs and is tailored to the process in your organization.

There are many good approaches to job analysis Most of them involve some type of general experience questionnaire and the observation and interview of selected employees doing their job. It is important to make sure that the employees have a good underst'nding of what is being done and why job related questions are being asked. Employees quickly become skeptical and concerned when a lot of questions are asked about their job and how it is performed. The cooperation given by the staff in programs such as this will be much better and with a more positive attitude if the reasons are first clearly explained.

Under each of the process steps the major skills, actions, and knowledge are identified that are important in accomplishing the step in the DP organization. Try to refine these major items as much as possible so they will be concise yet descriptive of the responsibility. Sometimes it is difficult to express all the items purely in the terms of skills. For this reason be practical and realistic about the results by using a mixture of skills, actions, and knowledge. This approach seems to render a better level of definition.

Figure 6 shows how this expansion can be accomplished. Keep in mind that we are talking about the application development function of the DP organization and the associated programmer and analyst staff. For this reason the expansion shown is for the first five steps of the process.

The technical strength definitions below show how knowledge levels can be identified. Once again these definitions should be prepared with the requirements of the organization in mind. There must be an adequate range in the knowledge levels. With programmers and analysts it is especially desirable to be able to identify whether they can explain a concept and apply it as well.

Technical Strength Definitions

0 has no knowledge
1 understands capability and applicability
2 can use technique effectively
3 can maintain, modify and create new features
4 can synthesize and evaluate to reduce or eliminate analysis, machine runs, development or tests based on his wide knowledge of alternatives and depth of experience.

You will notice that these levels are primarily for programmers and analysts so that we may continue to expand our definition of the application development function.

All the data that is needed to put together a summary skills matrix of a selected DP organization should now be available. The data includes:

1. Definition of the steps in the DP process to meet the needs of the business.
2. Definitions of job responsibilities to meet the process requirements.
3. Association of these job responsibilities with defined jobs.
4. Identification of the level and possibly the title of each of the defined jobs.
5. List of the major relevant skills, actions, and knowledge to meet the defined job responsibilities.
6. Define the strength levels of the skills and knowledge requirements.

A summary skills matrix can now be constructed by inserting the strength indicators required for each defined activity of each process step. The strength indicators are placed in the appropriate job title or level column.

Figure 7 shows how the technical strength indicators are added to the System Design step of the process. The strength level definition and the job level to which it is associated is determined primarily from the data obtained as a result of the job analysis and survey of the staff's skills.

When the summary skills matrix is constructed for all members of the DP organization, it becomes an invaluable management tool in planning and obtaining the required skill mix. The skills matrix also acts as a key index in planning tailored education for the staff. This can lead to significant savings in education expenses to a company.

Using the summary skills matrix and data from the job analysis and knowledge surveys, a trained industrial educator can determine what additional knowledge and skills are needed to enhance that already possessed by the staff. The identification of the missing skills and knowledge is the foundation on which future training

Name:
Business System Design

Duration:
Five days

Audience:
Those with the responsibility for the detailed design of a business system.

Prerequisites:
Experience in program development and testing or successful completion of the development and testing curriculum.

Objectives:
Upon successful completion of this course, the student will be able to:
Develop cost estimates and a work plan for scheduling and controlling the work through the design phase.
Define the record content, file organization, storage requirements and access methods.
Define data editing and validation rules, controls and audit trails.
Design the necessary forms and reports.
Complete the detailed documentation of the system, specifying the input, output, processing and file requirements of each program.
Prepare the systems test plan.

*Topics**	*Hours*
Systems design cost estimating planning and scheduling	2
Detailed record design, file organization and access method specification	4
Editing, validation, controls and audit trails	2
Forms and reports design	2
Procedure writing	2
Systems test planning, test data and file preparation	4
Program specification and documentation	4
Estimating programming development time	2
Case study	8
Total	30

* These topics teach to the 2 and 3 technical strength levels.

Figure 9—Course specification

can be based. Much of this needed training can be accomplished on the job or through self-study. Some of it will require more formal classroom education. In any case, the education will be tailored toward the business needs and requirements since it is based on a foundation geared toward that same goal.

Regardless of whether the training program required is going to be developed in-house or contracted out to an education company or computer manufacturing company, it is good to have at least a general specification of the type of training needed.

Using the related job responsibilities boxed in on Figure 8, a training specification can be written. Job responsibilities that are related should be clustered together so that eventually a series of topics or modules can be defined to teach the relevant subjects. Related topics, in turn, are normally grouped together to form a course.

The course specification shown in Figure 9 is one designed to teach to the 2 and 3 strength level on all the job responsibilities boxed in Figure 8. Notice that the course specification identifies the most pertinent facts about the training needed. The audience, prerequisites and a clear definition of the objectives are the most important items. Individual topics that make up the course and the approximate duration of each can be included when they are needed. The purpose for using a duration on each topic is to give some indication of the depth that topic covers.

I said at the beginning of this paper that surveys show that the greatest expenditure that a data processing organization has is for its human resource. Actually it is the knowledge that is being purchased. This statement applies equally as well to most all human resource requirements in industry. In Peter F. Drucker's recent book, *The Age of Discontinuity*, he treats knowledge as one of the four discontinuities. Mr. Drucker says, "Knowledge, during the last few decades, has become the central capital, cost center and the crucial resource of the economy."

Since knowledge is a very valuable and expensive commodity it should be treated as such. Planned education according to the needs of a business will impart knowledge where and when it is required.

Modular training—A new emphasis

by ROGER W. KLEFFMAN

United Air Lines
Englewood, Colorado

INTRODUCTION

During 1970 and early 1971, United Air Lines and IBM implemented a PARS-based (Programmed Airline Reservations System) reservations system. This system is written in Assembly Language Coding for the IBM 360/65. The software is special purpose, dedicated to providing very fast response to remote airline reservations inquiries including: schedules and availability between cities, passenger name data, and related information. At the start of the project, there were virtually no United personnel with PARS experience, and the implementation phase included the development of a comprehensive training program. The principle objectives of this program, over and above that of preparing United programmers and operators for participation in the project, included the capability to be completely self-sufficient for PARS training at United, the ability to administer training as personnel became available to the project, and the desire to maintain a minimal staff of professional trainers.

To achieve these objectives 12 special courses were developed using the guidelines described as modular training. These courses, developed by IBM for United, were according to specific guidelines and standards. The standards emphasize that courses should be developed in short, self-contained parts, bring the student into direct contact with source information and use other media (e.g., video tapes, audio tapes, slides) to amplify material being presented to the student. This assists the student in advancing at an individual pace, and encourages self-sufficiency.

The basis for this paper is that an in-house training program can be developed which is self-sufficient. The need for outside training may exist for course development. However, generally development can be accomplished by line personnel with technical competance designing training courses in accordance with well-defined standards and objectives. Once developed, it is possible to administer the training on an "as needed" basis, and to accomplish training on second or third shift, or even on a part-time basis.

The balance of the paper presents the details relating to this particular project. This includes principles of course development, course descriptions, and the required training sequence for positions on the project. It should be stressed that no really new principles or techniques are offered, only a new emphasis or applying some old methods (and not so old, such as video modules) is advocated here.

MODULAR TRAINING

Modular training is the term used by United to describe training in terms of courses divided into specified units. A course is defined as: a set of objectives that should be achieved by a target audience together with training material, a time duration for the course, and a plan for accomplishing the training. Training courses are tutorial for programmers and operators on this project in that training is viewed as the transfer of information necessary for the student to carry out job responsibilities in a defined area. This is in contrast to training in a seminar environment by means of exchange of ideas among participants. Under the modular approach, the responsibilities for programming and computer operator positions have been keyed to the courses. Each position has an identified sequence of training courses through which a student progresses prior to full-time work assignment.

Under this modular training approach, each course is administered to a student by an advisor, an experienced person with successful completion of the course or with extensive experience within that area. The governing elements of the course are a student guide and an advisor's guide.

The student guide is a document which coordinates the student's progression through training material. It is designed to give the student the information

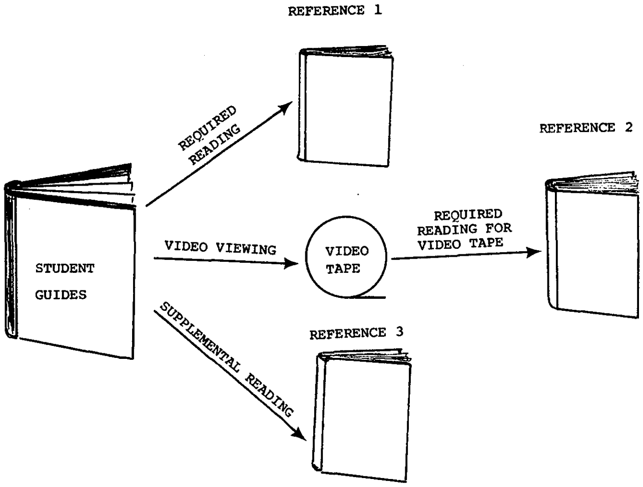


Figure 1—Student guide

necessary for a thorough understanding of the subject, as illustrated by Figure 1.

The intent of the student guide is to direct the student through applicable training material in a given order. This includes readings in references not contained in the guide itself (technical manuals, professional journals), or viewing video tapes (including local productions keyed to specific elements of the project), as well as utilizing the guide itself for tutorial material.

In particular the video tapes are a valuable means for producing training material. Use of video tapes is to deve'op standard, lecture-type modules and eliminates the requirement for an instructor to repeatedly give the same stand up lectures. Video is also useful in explaining standards and procedures, which can be presented to the student as dictated by students responsibilities to the project. The advisor, however, must always be available to answer questions or exchange dialogue with the student on the basis of the information in the video tape.

By design the student guide cannot and does not function alone as a self-sufficient course, such as in programmed instruction. The advisor is a necessary and vital part of the course in modular training.

The format of the student guide includes:

A. Introductory Section
   1. Course description and intended audience.
   2. Course objectives.
   3. Prerequisite training and/or related experience.
   4. Recommended duration of the course.
   5. Advisor/student relationship.

B. Reference Material
   This section identifies all other manuals and documents needed in the course.

C. Detailed Assignments
   This section presents the series of steps which lead the student through the material. It includes reading assignments, identifies lectures, designates participation in workshops, and tests to determine how well the student is learning and applying the material. In courses involving programming, actual coding experience is accomplished by assignments which are processed on the computer.

A history log is maintained for each course containing questions raised by students during the course. This information is used to develop additional explanatory material, quizzes, or more detailed references to aid the student in acquiring a better understanding of the subject.

COURSE ADVISOR

The second key element in modular training is the Course Advisor. By design the course can be administered to a single person, to a formal class of students, or to several students who have started the course at different times. Except for formal classes, the advisor performs this roll on a part time basis. As a result, line programmers can function as advisors as well as having normal job responsibilities.

Under United's approach, only one person devotes a full-time effort as the advisor for basic courses, and line programmers act as advisors for advanced courses. Each manager furnishes the advisor for advanced courses for students who will receive assignments in that manager's area. Only senior, experienced personnel are used as advisors. This represents one of their job responsibilities, and is a means of providing further experience in the management aspects of dealing with people.

The advisor has three primary responsibilities:

1. To serve as a point of reference for any questions the student may have on the course material. The advisor may not be able to answer every question, and may have to get the answer from another source. In any case, the response is on a timely basis.
2. To administer quizzes and exams to the student as required in the course sequence. The results are always reviewed with the student to insure complete understanding.

3. To emphasize the most pertinent parts of the course relative to the student's work area, and to evaluate the student's understanding of the course. Here, the advisor's function is to tailor the course to the student's needs. The advisor takes the initiative in discussing the important aspects of the course instead of passively waiting for the student to ask questions. By probing the student's thought process the advisor can fill in any areas of weakness and correct misunderstanding. Normally, the advisor spends ¼ to ½ of the student's assigned course time, depending on the advisor's familiarity with the course and the extent of the student questions. For example, if a student is training on a half-time basis, the advisor can expect to spend ⅛ to ¼ of that time in an advisory capacity. The advisor also provides up-to-date technical information on the system, which minimizes the problems associated with technical obsolescence of training material.

CAREER PATHS AND COURSE PROGRESSION

The United PARS project provides training for programmers, data base personnel and computer operator personnel. This section describes each job and the training sequence required.

There are seven areas of programming responsibility:

1. Application Programmer
   Develops programs in a real-time environment which accomplish the airline reservations functions. These include display of airline schedules, availability of seats inventory control, flight information, and programmatic exchange of information with other airline reservations systems. Programming is done in ALC, the assembly language used for coding on the IBM 360 series.
2. Schedule Change Programming
   Includes an understanding of batch processing as well as real-time programming, and has significant impact on many system data records. These include modifications to flight schedule information and development of information used to reaccommodate passengers. Programming in ALC under DOS and OS is included.
3. System Support Programmer
   Has responsibility for the generation and maintenance of systems, both OS and DOS, PARS off-line support, together with file maintenance and management reports. Programming languages include P L/I and FORTRAN.
4. Control Program Programmer
   Has assignments in input/output for disk and tape, CPU scheduling, queue processing for input, ready and deferred lists, interrupt handling and on-line data collection.
5. Communications Programmer
   Is involved with control and coordination of communications between remote sites with CRT terminals as well as with the other computer systems. There is heavy involvement with hardware such as IBM 2703, WE201B modems or equivalent, IBM 2948, and the IBM 2915 or equivalents.
6. Performance Evaluation Programmer
   Analyzes the PARS system performance by measuring and interpreting system characteristics including input/output message size, program core residency time and execution time, file accesses and queueing. Simulates the system by means of model(s) and predicts system performance relative to proposed modification.
7. Coverage Programmer
   Has primary responsibility for the diagnosis and immediate correction of on-line software malfunctions, as well as monitoring on-line system software integrity. Programming occupies a relatively small amount of time.

Two Data Base jobs pertain to information records in both the on-line and off-line systems. Programming experience is not required for these jobs:

1. Data Base Specialist
   Has the responsibility for the integrity of all passenger information in the on-line system. This can involve restructing parts of the data base as a result of airline rescheduling as well as corrections to erroneous on-line information.
2. Data Base Coordinator
   Has responsibility for maintaining airline schedule information, pilot data information and communications tables. These are processed in the off-line system.

Computer Operations involves three tasks:

1. Data Control Clerk
   Receives and monitors off-line processing jobs, maintains tape and disk libraries, and JCL card decks for off-line jobs.
2. Computer Equipment Operator
   Operates all equipment in the off-line-system

		PROGRAMMER							DATA BASE		OPERATIONS		
		APPLICATION PROGRAMMER	SCHEDULE CHANGE PROGRAMMER	SYSTEM SUPPORT PROGRAMMER	CONTROL PROGRAM PROGRAMMER	COMMUNICATIONS PROGRAMMER	PERFORMANCE EVALUATION PROGRAMMER	COVERAGE PROGRAMMER	DATA BASE SPECIALIST	DATA BASE COORDINATOR	DATA CONTROL CLERK	COMPUTER EQUIPMENT OPERATOR	CONSOLE OPERATOR
360 TRAINING	COMPUTING SYSTEMS FUNDAMENTALS										R1	R1	R1
	INTRODUCTION TO S/360	R1	R1	R1	R1	R1	R1	R1	R1	D	R2	R2	R2
	ASSEMBLER LANGUAGE CODING	R2	R2	R2	R2	R2	R2	R2	D				
	SYSTEM OPERATOR TRAINING-DOS										R3	R3	R3
	INTRODUCTION TO OS										R5	R8	R8
	SYSTEM OPERATOR TRAINING-OS										R6	R9	R9
	OS ADVANCED TRAINING											R10	R10
	DOS/OS JOB CONTROL LANGUAGE		R3	R3			D	D					
	TOTAL TRAINING DAYS FOR REQUIRED COURSES	15	18	18	15	15	15	15	5	-	21	25	25
PARS TRAINING	PARS SYSTEM DESCRIPTION	R3	R4	R4	R3	R3	R3	R3	R2	D	R4	R4	R4
	APPLICATIONS PROGRAM	R4	R5	R5*	R4	R4	R4	R4	D				
	AIRLINE PROGRAMMER ENVIRONMENT	R5	R6	R6*	R5	R5	R5	R5	D				
	AIRLINE CONTROL PROGRAM FUNDAMENTALS			R7*	R6	R6	R6	R6					
	AIRLINE CONTROL PROGRAM INTERNALS			R8*	R7	R7	R7	R7					
	SCHEDULE CHANGE-INPUT DATA PREPARATION		D							R1			
	OFF-LINE SCHEDULE CHANGE		R7						D	D			
	ON-LINE SCHEDULE CHANGE		R8					R8	R3				
	COMMUNICATIONS					R8		R9					
	DATA COLLECTION				D	R9	R8	R10					
	DATA BASE MANAGEMENT			R9*				R11					
	SYSTEM OPERATOR TRAINING							D			D	R5	R5
	TOTAL TRAINING DAYS FOR REQUIRED COURSES	25	35	50	45	55	50	80	10	3	5	10	10

R = Required
D = Desirable

Subscript indicates course sequence number

***Not required for all System Support Programmers**

Figure 2—Career path training sequence

and assists in monitoring the on-line system activities.

3. Console Operator
Has responsibility for operating the hardware in the on-line system. Monitors and controls system configuration, and applies corrective action for hardware failures.

The training required for these positions is summarized in Figure 2. IBM/360 training is also shown in presenting the training progressions. The total training days associated with the IBM/360 training and the PARS training identify the time needed before an individual is ready to accomplish productive work.

For example, a programmer with no 360 computer background would require both IBM/360 and PARS training. For applications programming, 40 training days would be required. If the programmer already had IBM/360 experience and no PARS experience, 25 training days would be required. The description of PARS training courses is given in the Appendix. Both the training results and the training duration are good as evidenced by high comprehension rate and mininal training period.

SUMMARY

As a part of the implementation of a PARS system, United Air Lines has developed a comprehensive training program using the modular method with three objectives:

1. United is self-sufficient for PARS training (Note: PARS training requires IBM 360 knowledge as a prerequisite).
2. Training is administered as people are available.
3. Training is accomplished with a minimal professional training staff.

Twelve courses were developed to achieve these objectives for the different job responsibilities within the project. More than sixty students were successfully trained with this method.

An advisor is a key element in administering modular training. The advisor is responsible for tailoring the basic course to reflect the content of the student's assigned area, and to monitor the student's progress. Drawn from the line organization, the advisor is proficient in the assigned area and monitors the student's progress by answering questions, administering quizzes, and probing the student's understanding of the subiect matter.

Modular training with professionally competent personnel acting as advisors eliminates the need for formal classes, and training can be administered on an "as available" basis, although a class can easily be accommodated with this method. In addition, the advisors have line responsibilities and thus the need for a large, dedicated training staff is eliminated. Currently, at United one person has full-time basic training responsibility for the 70-80 project programming staff.

Advanced training advisors are furnished by the area to which the student is assigned.

Generally only senior persons are advisors and represent a part of the responsibilities of a senior position. Being an advisor usually affords first exposure to working with people in a management mode, which is good experience if further advancement along supervisory lines is desired.

In particular, however, it was found that:

1. An advisor is indispensable to the program, and cannot be passive. The advisor must actively monitor and encourage the student's progress. Experience has shown this method has favorable results.
2. Prior to starting the training, there should be a review meeting between the student, the assigned manager and the advisor. The review will include the sequence and duration of the courses for the student, the approximate expected progress (relative to whether the student will be in training full-time or part-time), and the role and responsibility of the advisor(s).
3. Practical problems, and quizzes which reflect the course objectives are essential, especially for programming courses. A simulation of the real environment is necessary for coding and debugging problems. The course content must be tutorial and serve as a transition into the work environment.
4. Insofar as possible, training, especially the advanced training, should be on a part-time basis. This gives the student a better opportunity to assimilate the many new concepts and terminology as training exposes them. In fact, terminology is a major factor on this project, because the student is not only using EDP terms, but also the special vocabularies associated with airlines and with the PARS system.
5. Update of course content caused by system change is best accomplished by line personnel. In this case they act as advisors but without students, and have the responsibility for updating the applicable course content.
6. The student guide/advisor functions best in a non-development environment. In a totally developmental environment, scheduling is a problem. Estimates of required program development time always seem to neglect factors which were not expected to occur (sickness, machine downtime, etc.) In either case, the line must be staffed to accomplish both programming and training requirements.

Overall, modular training has worked reasonably well for the United Airlines PARS project. The primary objectives have been satisfied by a minimal professional training staff and utilization of line personnel in the training function. The advisory responsibility takes time, however, and an advisor's schedule must include time for training in addition to normal line responsibilities if modular training is to function properly.

This same method should also be able to work for other projects. A student guide need not necessarily be a reference unto itself, it may be a series of references to other manuals, documents, video tape, et al. Video tapes, in particular, afford the means of augumenting course material and making it available for training around the clock. There currently exists a dearth of organized information and material which can be used to construct a course. There is no prophet for rewriting a document which can be used in its present format with directions as to applicability. Both the student guide and the advisor furnish this direction to the material and provide tailoring specific to requirements.

APPENDIX A

Course descriptions

This Appendix presents a description of the 12 courses that were developed at United Air Lines. These specific courses are applicable only to a relatively small group of PARS users.

It should be noted that because the courses are tutorial, quizzes form an important part of the course, being geared to the course objectives. The student is expected to achieve certain defined objectives, and the quiz results represent, in part, how well the material has been learned, as well as performance evaluation.

Training in PARS requires a knowledge of the IBM 360 hardware and Assembly Language Coding (ALC). These are prerequisites for all the PARS Programming Courses. Commercial video taped courses, augmented by material particular to United's needs, satisfy these requirements.

The PARS courses are divided into two sets; a basic set which all programmers attend, and the advanced courses, which are attended on an "as needed" basis. The first three courses listed below are the members of the basic set; all others belong to the advanced set for programming except the operator course, which is independent. Sixteen video tapes have been produced by United to augment the basic PARS training. These were produced based on an evaluation of the information in the history log, an invaluable tool for developing training modules such as programming techniques, project procedures, and the like. In addition, the video modules are always available to the student on an around the clock basis, thus helping to free training from the prime shift dependency.

1. PARS System Description—3 days
   This course presents the basic PARS hardware and software descriptions, as well as concepts and terminology associated with the airline industry as well as PARS system. Emphasis is placed on providing the fundamental information concepts and terminology needed in other courses.
2. Applications Programs—5 days
   This course presents the functions of all PARS application program packages including programs within each package, and describes the data records required by each program explaining how these records are accessed and data is manipulated. It relates airline reservations agent actions and needs to the manipulation of the applications programs and the data records.
3. Airlines Programmer Environment—17 days
   This course is divided into three major areas corresponding to the types of programmer activities in this real-time PARS environment:
   1. Program Development
      The availability and use of programmers tools such as: entry control block, macros, and conventions.
   2. Program Maintenance
      The PARS Program Library System, the steps involved in entering programs, the inputs necessary for program maintenance.
   3. Program Testing
      The concepts and procedures used in testing, in generating test inputs and in interpreting test results.

   The course environment is designed to resemble, as closely as possible, the expected environment in the development group. The subjects discussed at length are those dealing directly with the job of application programming.
   Upon completion, the student is capable of productive assignment in applications programming within the PARS environment.
4. ACP Fundamentals—5 days
   This course introduces the programming techniques such as queuing, priorities, message flows, and related items used by the Control Program.

Upon completion the student is able to relate the functions of the ACP program to his present assignment.

5. ACP Internals—15 days
Airline Control Program Internals is concerned with program listing level discussion, by topic, of each of the functional units of the airline CP. This includes communications, hardware input/output, and services and facilities provided to application programs.
Upon completion the student is prepared to accept assignment in any functional area of the Control Program. The student will lack only what can be gained through experience.
6. Schedule Change Input Data Preparation—3 days
Editing and validation of modified schedule information is the main objective of this course. Upon completion the student is able to prepare the input data required for Off-line Schedule Change.
7. Off-Line Schedule Change—5 days
Topics covered include concepts of off-line schedule change; off-line schedule change input and processing, and a detailed analysis of the off-line schedule change programs.

   Upcn completion the student is able to communicate in the language of schedule change with non-programmers and other PARS Application Programmers, use to use the appropriate documents to extract the details of the schedule change programs assigned as the student's responsibility.
8. On-Line Schedule Change—5 days
This course presents the functions of all the on-line schedule change programs, defines the data records affected by the on-line schedule change and describes how they are affected, explains the four logic phases of the on-line schedule change, and relates the programs and data records involved to each phase.
9. Communications—5 days
This course presents PARS Communications in general, and specifically as applied in United's PARS System. This system exchanges communications with two other United Air Lines On-line Computer Systems. In addition PARS Communications is covered from the functional level to a detailed program level which describes the interrelationship of the programs, tables and data records involved.
10. Data Collection—5 days
This course describes the functions of the Data Collection Programs and defines the facilities provided by these programs, explaining how system, file, program and message variables are collected on-line, and reduced off-line. It identifies listing variables which are out of line from data collection reports and determines from these variables how system performance can be improved.
11. Data Base Management—20 days
Data Base Management covers four major file support functions and data base generation. These include: Pool Directory Generatiɔn and Maintenance, Recoup, Capture/Restore, Fixed File Reorganization and System Test Compiler.
12. System Operator Training—5 days
This course describes, in general terms, a typical PARS hardware configuration and the PARS software system. In addition performance of the following operator tasks is explained: program library maintenance and update, testing, system load, all other off-line and on-line operator activities, interpretation of error messages, and use of functional messages appropriate to a given job.

Computer training, present and future

by GORDON A. SMITH

Jet Propulsion Laboratory
Pasadena, California

INTRODUCTION

Before beginning this discussion, I would like to make a few remarks concerning the Jet Propulsion Laboratory (JPL). The Jet Propulsion Laboratory is responsible to the National Aeronautics and Space Administration (NASA) for unmanned missions in deep space. A worldwide tracking network encompasses Goldstone, California; Madrid, Spain; Johannesburg, South Africa; and Woomera and Tidbinbilla, Australia. In addition to the deep space missions, there are a great number of scientifically oriented projects throughout JPL. The Laboratory has approximately 4,100 employees, as well as some support contractor personnel.

All levels of personnel in JPL, as well as contractor operations personnel, are involved with the educational programs, starting with "Introduction to Computing" through management training in the data processing field.

Within almost every government organization, and certainly within every NASA facility, a considerable workload rests within the administrative and financial operations. These are nonscientific endeavors which hopefully will lead toward improved management information system controls. A commercial enterprise, financial operations organization has a number of similarities.

To begin with, I have prejudices against classrooms of stand-up instruction only, or classrooms of video-taped lecturers only, or instruction in any single medium. Therefore, I will spend a few moments reviewing some of the products of educational firms today leading toward a multimedia approach to the education and training of commercial programmers, systems analysts, or obviously, into some of the scientific fields.

Let me cite a few examples of ideas that have come out of some commercial firms. For instance, there is one firm in Ohio that has developed a course titled "ADP Terminology." This course consists of 16 audio tape cassettes and a small workbook for those who want to become familiar with data processing terms. One of the major banks in Arizona has required its 132 branch managers to review this course while traveling to and from work. It is also an excellent device to augment stand-up instruction for a new student. Several firms today have canned video-taped courses along with workbooks as a self-instructional means to replace the very widespread utilization of PI or programmed instruction textbooks. To my way of thinking, there is nothing more boring, and less motivating, than sitting down with a workbook, and saying I am going to learn by myself and teach myself something, whether it is data processing or any other area of interest.

Another firm, in addition to its video taped courses in basic computer programming—COBOL, FORTRAN, and so forth—has a considerable library of animated motion picture films, along with associated workbooks and other materials as a self-teaching means in support of standardized automatic data processing (ADP) curricula. Some of the films are excellent, and, in some cases, we have utilized them to augment our classroom course "Introduction to Computer Technology." A firm in Illinois has developed a systems analysis course, as well as courses in operating systems (OS) with a thorough interaction between video tape, audio tapes, and workbooks—none of the media being used for more than 15 minutes at a time. There is a constant transfer from one medium to another which provides not only motivation but reinforcement. More of the physical senses are used in the absorption of educational material. Of course, motion picture films are also used; films themselves are a bit more cumbersome, but if intermixed with other media, may become a viable means of instructional transfer.

I believe that audio tapes, video tapes, motion picture films, and animated cartoons can effectively be used in a learning carrel. A carrel is nothing more than an enlarged desk with various devices in it; one can

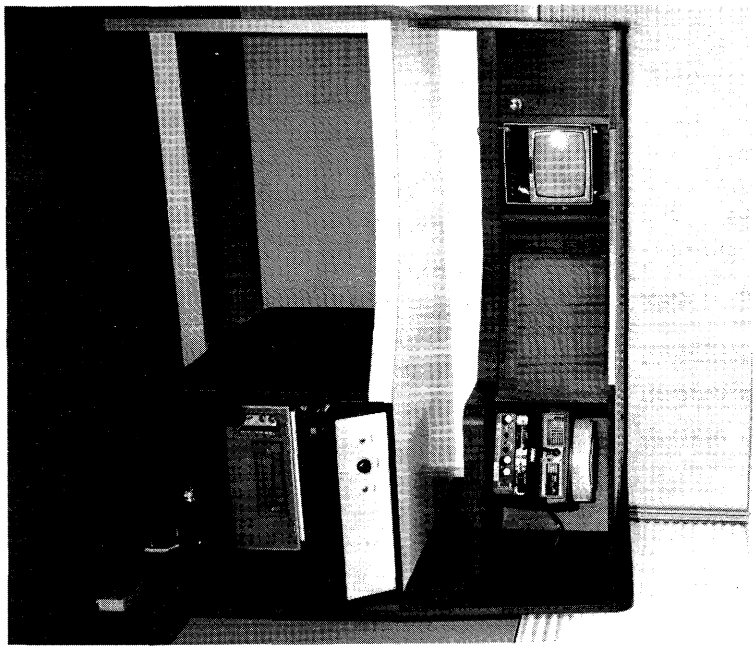


Figure 1—JPL training carrel

instruct himself at his own speed, and hopefully, improve his motivation if he has a goal in mind (see Figure 1). This type of self-instruction is extremely worthwhile if it is also reinforced by a senior programmer or analyst. The multimedia approach will create greater motivation.

For a moment, let's look at the benefits. In classroom training, the teacher is usually responsible for maintaining a norm at the average class level. Very often the faster student becomes quite bored because he is ahead of the group, whereas the slower student is having a problem keeping up, and thus becomes rather discouraged. Most schools in the data processing field require the student to write, run, and debug test programs on whatever type of equipment that school may possess. There is a danger at times if the "school" has only the equipment of one manufacturer and is not able to provide good classroom direction where students can effectively learn the differences between equipment of various manufacturers—either small-scale computers or large-scale systems.

A list of the computer courses available at the Jet Propulsion Laboratory, along with the computer selection procedure, is given in Appendix A; a set of computer-related job descriptions from the *Computer Education Directory* published by Edutronics Systems International, Inc., Los Angeles, Calif., is presented in Appendix B.

COMPUTER-ASSISTED INSTRUCTION

Computer-assisted instruction, or CAI, is a relatively new medium to augment the other media I have discussed; however, I am certain all of us will be influenced by it over the next few years. I feel we are missing a very worthwhile and cost-effective tool. Helen Lakin at the University of Wisconsin noted there are 1,264 program descriptions in the publication *Index to Computer-Assisted Instruction.* These descriptions are from sources in the United States, Canada, Europe, and Japan, representing elementary and secondary school systems, colleges, and universities that have complete CAI courses for credit. As an example, the University of Southern California recently instituted a complete course in Programming Language I by CAI for three units of credit. Many universities are using CAI as a self-teaching method very successfully for both direct teaching and remedial education. In none of these situations does it mean that there is not a teacher available to answer questions or at least to cover the highlights that may be misunderstood. In the government area, certainly throughout the Department of Defense, there are any number of programs that are offshoots of simulation programs as well as specific courses in CAI. Many of the programs, however, are tailormade to the particular command or to the organization.

It is extremely important that data processing organizations thoroughly evaluate available software systems in today's market, whether they are proprietary or in the public domain. For the programmer or systems analyst, where it can be shown to be cost-effective, there is no better system than one with immediate feedback. From a motivational standpoint, the one-to-one relationship without outside distraction can be made a very viable and successful means of instruction in this field. It seems strange that there are not more schools or private firms teaching programming, systems analysis, or data processing management on a computer terminal.

Briefly, from the principal title of the session, let's consider the capability of the computer in CAI. In August 1970, a test project was instituted in the Communications Electronics School Battalion at the Marine Corps Recruit Depot in San Diego, California. Twelve terminals were connected by data link to a UNIVAC 1108 computer some 75 miles away. A stand-up class was run concurrently with the CAI classes. Individuals were psychologically matched, and each student proceeded through the course of instruction at his own pace. Of considerable importance, the Marine instructors wrote their own courses for the computer after only two weeks of instruction from the manufacturer. A system of positive or negative reinforcement of learning experience was presented to the student immediately. The immediate benefits of the teaching experience were the objectivity, data collection and reduction, and standardization of instruction. Cathode ray tube terminals were employed. The results demonstrated substantially increased learning in the CAI class; and subsequent to the test, far greater retention was achieved. This demonstration has convinced Headquarters Marine Corps to initiate formal schools in CAI at the Marine Corps Station at Twenty-Nine Palms, California. Sixty terminals will be employed, course time will be reduced by 20 percent, and instructor requirements by 75 percent of instructor manpower.

In the universities, there is a strong tendency to tie CAI courses with other portions of the curricula. There is a major tendency, of course, to deal primarily with the fields of scientific uses of the computer, or, in other words, its applications to physics, chemistry, biology, genetics, etc., rather than to the straight business approaches; however, some universities are improving their curricula, from computer programming on into the area of management information systems. There is an excellent marriage between business management and the tools the computer can provide.

GOVERNMENT USE OF COMPUTERS

The government was the first major user of computers, in the late 1940s and early 1950s. Commercial data processing centers have become increasingly widespread. Subsequent to unbundling, the government agencies developed their own schools and separate series of grades within the Civil Service Commission to handle the various levels of computer programmers and systems analysts. One of the problems of government, as well as of all large business operations, is frequently a lack of communication among similar services and organizations. The Office of Management and Budgets in Washington is examining improvements in communication among similar elements of government, where both can utilize the same software, hardware, or systems approaches. Where there is coordination among government entities, cost savings are very substantial. Due to a low turnover, many large government entities also realize that their management personnel, as well as their programming personnel do not effectively utilize the tools available today.

Many of the leading computer manufacturers emphasize more now than before the need for improved management training which, in a sense, goes beyond the scope of this present paper. However, when speaking of education for computer programmers and systems analysts, we must consider the people who are directing the needs for this education and training. These needs definitely stem, and should stem, from a better understanding by management of their own specific needs. When management understands these needs and can define them, then indeed they will do a much better job in hiring personnel. This will establish the type of training curricula these personnel should have had before being hired.

As a lesson from the government's interest in this field, communicating between commercial firms possessing similar interests can assist industry in setting standards or norms of personnel. In Appendix B are a number of what I feel to be excellent definitions of programmers and systems analysts. There is a need throughout the United States today for an improvement in semantics of all facets of ADP terminology in the entire field. Personnel firms should seek for a common understanding, thus setting some type of definition of various grades of programmers for the systems analysts. In large industries, of course, these have been indicated. However, they vary considerably from industry to industry. By an improvement in definition, understanding, and attainment, the individuals in this field should become more professional in their community.

THE IMPORTANCE OF PRETESTS

Obviously, any paper may reflect the bias of the individual writing that paper, who may promote those things that are the most successful in improving his responsibilities. Both with beginning programmers or systems analysts, or with those who wish to enter this field, I am a firm believer in computer pretests. Large schools have pretests of one type or another. However, when they are selling courses, their pretests may not be as restrictive as a pretest would be within my organization or within other organizations depending upon a profit motive.

Pretests can aid in determining a person's ability to absorb a particular instructional level. He takes the test, analyzes the test himself, and, from the results, he himself should be mature enough to state whether he has the ability to go ahead as a computer programmer, as a systems analyst, or in some other field of data processing, or whether his capability in fact rests somewhere else, and not in the ADP field at all. Many of the personnel with whom I have been associated who have taken these pretests have suddenly decided they should not take FORTRAN or COBOL, but rather should go back into a program we have titled "Introduction to Computers," or, in other words, the basic groundwork—"What is a computer, what does it do, how does it operate, how does it function, and what are the various fields to which it could be applied?" The next step (shown in Figure A-1) analyzes his particular functional responsibilities, and, from there, determines the type of course toward which he should direct himself: in other words, what computing system should he be addressing? This last step is extremely important, since a person who is completely educated in one type of equipment, and then is thrown into another organization with an entirely different type of equipment, normally has some problems adjusting. This can be prevented if his training is directed toward the type of equipment he is going to use in the first place. Within an organization, a person's immediate supervisor is responsible for determining whether company time should be devoted to such courses or whether the individual would improve himself by taking courses outside of his office hours.

FUTURE DEVELOPMENTS

Let's consider some probable areas where changes in technology should considerably influence ADP education in the next decade. Increasing capability in the multiprogramming aspects of large-scale computers will permit a more economical approach toward computer-assisted instruction. Initially, however, it should be treated primarily as an augmentation or reinforcement of courses rather than as the stand-alone desire of some CAI enthusiasts. There will be an increasing need for studies into the motivational aspects of all levels of individuals, whether students are from grammar schools or high schools, or are personnel in government or industry.

According to the ETV Newsletter of July 26, 1971, as an example of the future development (and one that I believe will be widespread some years from now), the MITRE Corporation of Bedford, Massachusetts, is conducting a feasibility study utilizing TV/CAI through cable TV distribution. In this instance, both the voice and the still pictures are transmitted via microwave and on cable TV to homes equipped with special terminal and TV set combinations. The company is using its time-shared interactive computer-controlled information television (TICCIT) system to display selective material on individual TV sets. Thus, 600 TV sets can receive separate information provided by the computer at a typical rate of one frame every 10 seconds. The viewer is then in a position to communicate back to the computer through a 12-button telephone. The overall software for the system is designed under the sponsorship of the National Science Foundation, which can provide individualized educational courses for home and school use. Over courses that have a broad-scale interest in the future, therefore, it is quite likely that up to 10,000 TV sets in one local area could conceivably be tied into one major system with a number of different programs being selectively addressed.

Microcomputers of the future, costing as low as \$2 to \$3 K with keyboard and tape cassette units and a small TV light display, could be utilized by a number of commercial and governmental operations where specific instruction is required.* I believe the cassette will play a large part in the future.

Along with educational programs, we consider the terminal hardware development. Keyboards have varied from the standard typewriter keyboard to color-coded keys for some special studies. It is possible the lightweight, touch-sensitive surfaces instead of keyboards may be utilized. Common phrases that could have a general application to a number of educational uses may be designed into either system.

Pictorial or handwritten input may be one area of the future, as well as the very early development of voice input (Culler Harrison, Inc., in Goleta, California, and the Bell Telephone Labs), and, lastly, one I am certain a number of you have seen demonstrated, the plasma array terminals developed by Dr. Donald L. Bitzer of the University of Illinois in conjunction with Owens Illinois Glass Co. As some of you may recall, the view plate which replaces the video tube consists of three layers of glass with gold wires and an X/Y axis, imbedded in an electrolyte containing neon gas. The dots of stimulated neon gas, 3600 per square inch, may display pictures without computer regeneration, allowing both the computer and the user to modify the display. A number of these terminals are presently in manufacture by The Magnavox Co. for the University. I feel this type of approach will contribute a great deal to the ability of fairly small computers having a capability to relate through educational systems to a number of terminals.

The software field also has a major impact upon the future of education and training to systems analysts and programming personnel, in that language capa-

* F. W. Blackwell, *The Probable State of Computer Technology by 1980, With Some Implications for Education*, P-4693 R, The Rand Corporation, Santa Monica, Calif., Sept. 1971.

bility should certainly be on higher level than it is at present, including switchable microprograms, very large memories, associative memories, and other variables at an economical cost which could contribute considerably to a multi-user access. In a sense, with the technological changes, both in hardware and software, the aspect of the present programmer training should change markedly from the existing languages and the present direction. One of the interesting fallouts in teaching FORTRAN to a group of high school students on the West Coast recently resulted in improved ability of the students in their other courses. Apparently there is some relationship to the logical display of FORTRAN which overlapped into an improved understanding by these students in other subjects within their curricula.

Another aspect of the future is the multitude of libraries of application programs available to all facets of industry, government, and the educational community. With improved library maintenance and cross correlation, hopefully, many users of the future will be prevented from "reinventing the wheel" concept, and will be forced to thoroughly review the application programs available before they attempt to develop training programs on their own. Therefore, extensive markets exist for personnel to improve the development of such a compendium. Along with these application programs, however, and one of the aspects that is not as thoroughly covered today, is that of computer costs associated with each application program. This type of data will definitely have to be included in any compendium of library programs (similar to COSMIC,* which is operated today).

In communications, I feel many of us have seen limitations in our present telephone equipment; however, in the near future, firms similar to DATRAN, whose microwave towers will be completely devoted to data processing centers throughout the United States, as well as improvements in our present communication equipment and satellite interface, will aid considerably in the more economical aspects of computing, as well as the education and training in a CAI approach throughout major time-sharing networks.

In summary, there will be a considerable change in programming languages, major changes in terminals and hardware equipment, and a greater interest and acceptance of computers.

RECOMMENDATIONS

For the educational, governmental, or business organization involved with its own educational development, it is essential to seek a multimedia mode of educating personnel. Do not rely on self-instructional manuals for the individual to develop an ability on his own. Upon completion of internal or external schooling, maintain an open view to periodic reinforcement of whatever course the individual attends.

Whenever possible, seek or have developed pretests to determine the level of investment your organization would make with the individual. It is normally wiser to allow management personnel the prerogative for self-correcting to determine their level of instruction. I feel that senior and middle management are too often neglected in the general concepts of computer technology.

Be aware of the developments in the university and governmental areas in education and training as they may very often be obtained without cost to your organization. The NASA data bank (COSMIC) is an example of this.

Lastly, in returning to the major subject of this session, that of training of commercial programmers and systems analysts, a thorough review of the commercial school, in addition to the very excellent training offered by the major computer manufacturers, should be made. University or high school evening classes very often may be an economical advantage in this evaluation. Maintain an awareness of the development of CAI as it is being used in grammar schools, secondary schools, and universities. Most important of all, to the prospective programmer or analyst, be aware of the "state of the art" changes to prevent ccmplacency.

ACKNOWLEDGMENTS

This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under Contract No. NAS 7-100, sponsored by the National Aeronautics and Space Administration.

APPENDIX A—COMPUTER COURSES AT THE JET PROPULSION LABORATORY, CALIFORNIA INSTITUTE OF TECHNOLOGY

GENERAL INFORMATION

Definition of terms

Pretest. These are self-administered, self-graded tests whose purpose is to help the individual determine that

* The Computer Software Management and Information Center (COSMIC) is maintained for NASA by the University of Georgia and provides a national data bank of software programs developed and debugged by government agencies and in the public domain.

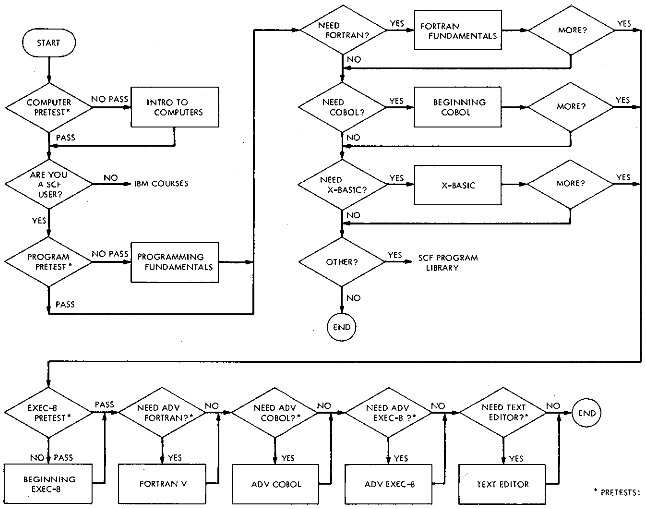
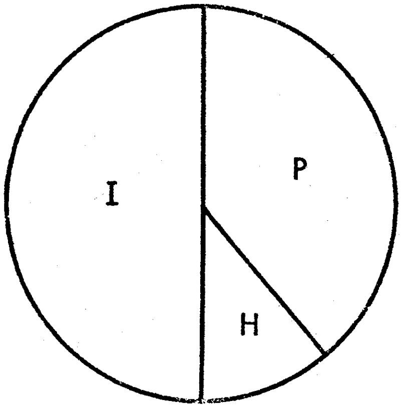


Figure A-1—JPL computer course selector

the course is correct and that the individual is neither underpowered nor overpowered for the course material. The pretest also provides a general feeling of course content.

Time distribution charts

Each pie chart indicates the approximate distribution of the total time of the course required both in and out of class. The symbol "I" refers to Instruction, "P" refers to Participation, and "H" refers to Homework.



Certification

Certificates of completion are issued and recorded in the personnel file. They are based either upon attendance or upon the demonstration of proficiency in the techniques or skills taught.

Survey

The courses were developed in response to the major needs expressed and measured through the Training Needs Survey.

Schematic

A diagram of the courses available is shown in Figure A-1.

INTRODUCTION TO COMPUTERS

For Whom: Those with little or no knowledge of computers. Quota limited.

Prerequisites: None.

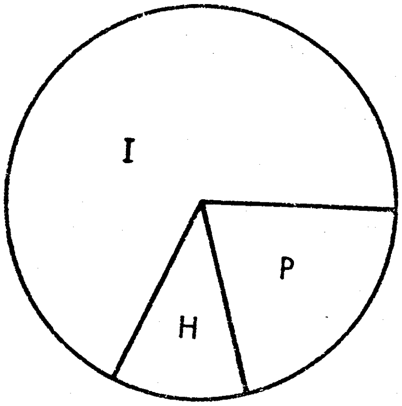
Description: Description and explanation of computer purposes, parts and functions, computer languages, programming concepts, and the computers at JPL.

Objectives: To gain a general understanding of computer-related technology and terminology.

Text Provided: Handouts.

Pretest: Recommended.

Time Distribution:



Instruction: Lectures with audio-visuals.
Participation: Discussion.
Homework: Simple problem assignments.

Schedule: Four 2-hour sessions; 2 weeks. A new class to be held each month during the year.

PROGRAMMING FUNDAMENTALS

For Whom: Those with no computer programming background who plan to take other courses in computer languages. Quota limited.

Prerequisites: None.

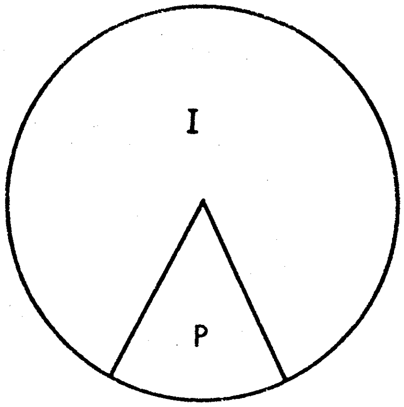
Description: Introduction to computer programming, its logic, elements, terminology, and procedures. Preview of FORTRAN, COBOL, and X-BASIC.

Objectives: To gain general awareness of computer programming concepts and languages.

Text Provided: None.

Pretest: Recommended.

Time Distribution:



Instruction: Live and video-taped lectures.
Participation: Discussion.
Homework: None.

Schedule: Four 2-hour sessions; 2 weeks. Classes to be held each month during the fiscal year.

FORTRAN FUNDAMENTALS

For Whom: Beginning FORTRAN programmers who will use the Scientific Computing Facility. Quota limited.

Prerequisites: "Programming Fundamentals" course.

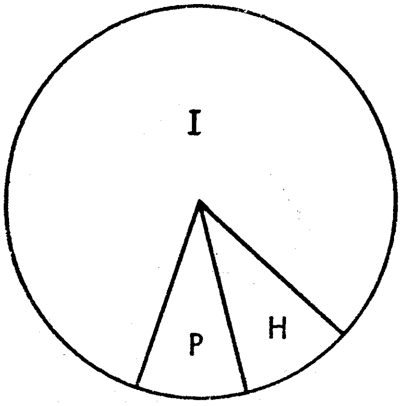
Description: Introduction to the elements of FORTRAN and the methods of writing programs in it. This course includes several illustrative problem assignments.

Objectives: To gain ability to write simple programs in FORTRAN.

Text Provided: FORTRAN IV primer—class notes.

Pretest: Recommended.

Time Distribution:



Instruction: Video-taped lectures.
Participation: Discussion.
Homework: Problem assignments.

Schedule: Twelve 2-hour sessions; 6 weeks. Classes to be given on a bimonthly basis.

FORTRAN V

For Whom: Scientific Computing Facility users. Quota limited.

Prerequisites: FORTRAN Fundamentals and "Beginning EXEC-8" courses and experience using the UNIVAC 1108.

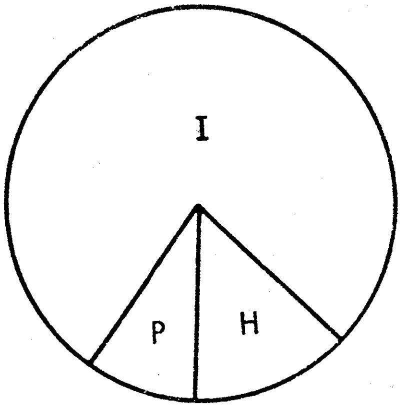
Description: Review of FORTRAN Fundamentals. Explanation of more extensive capabilities of FORTRAN for advanced programming technology.

Objectives: To gain ability to write sophisticated programs in FORTRAN.

Text Provided: FORTRAN V Reference manual.

Pretest: Recommended.

Time Distribution:



Instruction: Lectures.
Participation: Discussion.
Homework: Program assignments.

Schedule: Ten 2-hour sessions; 3½ weeks. Six to eight classes per fiscal year.

BEGINNING COBOL

For Whom: Those who will use the Administrative Computing Facilities. Quota limited.

Prerequisites: "Programming Fundamentals" and "Introduction to Computers" courses or equivalent.

Description: Introduction to the elements of COBOL and methods of writing programs. Several illustrative problem assignments are included.

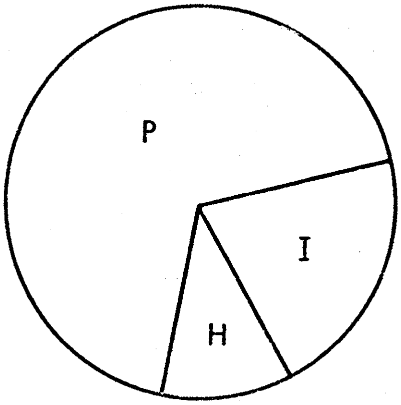
Objectives: To gain ability to write simple programs in COBOL.

Instructor: Self-study program with group discussion.

Text Provided: 1108 ANS COBOL—Programmers Reference Manual.

Pretest: None.

Time Distribution:



Instruction: Discussion leader and films.
Participation: Self-study.
Homework: Simple problem assignments.

Schedule: Four hours self-study approximately, plus 1-hour group discussion per week; 7 weeks. Four classes per year scheduled.

BEGINNING EXEC-8 CONTROL LANGUAGE

For Whom: Beginning users of the Scientific Computing Facility. Quota limited.

Prerequisites: Ability to use X-BASIC, FORTRAN, or COBOL.

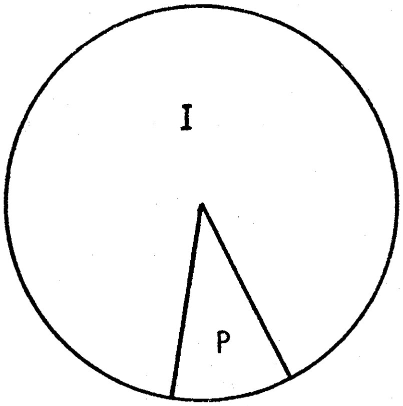
Description: "Introduction to EXEC-8," the Command Language of the UNIVAC 1108 System. Preparation of run streams, compilation of programs, assignment and manipulation of files.

Objectives: To gain ability to prepare simple jobs for running on the UNIVAC 1108.

Text Provided: EXEC-8 Student Handbook.

Pretest: None.

Time Distribution:



Instruction: Lecture.
Participation: Discussion.
Homework: None.

Schedule: Four 1½ hour sessions; 2 weeks; classes to be held each month.

EXEC-8 CONTROL LANGUAGE

For Whom: Scientific Computing Facility users.

Prerequisites: Extensive experience with the SCF computer system.

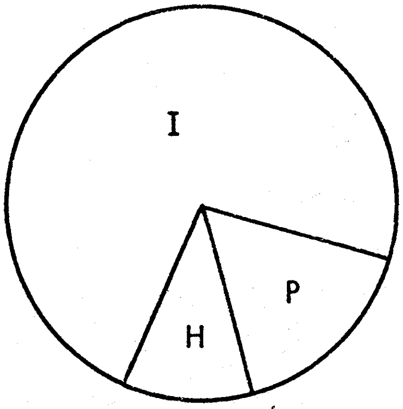
Description: Explanation of the full capabilities of the EXEC-8 Control Language, system file formats, and programming debugging aids and methods.

Objectives: To gain ability to prepare jobs for the UNIVAC 1108, utilizing the full capabilities of the system available to higher level language programmers.

Text Provided: EXEC-8 OPERATING SYSTEMS—Programmers Reference Manual.

Pretest: Recommended.

Time Distribution:



Instruction: Lecture.
Participation: Discussion.
Homework: Problem assignments.

Schedule: Ten 2-hour sessions; 5 weeks; four classes/year.

TEXT EDITOR FOR THE UNIVAC 1108

For Whom: Scientific Computing Facility users with access to conventional terminals. Quota limited.

Prerequisites: "Beginning EXEC-8 Control Language" course and experience in use of conversational terminals.

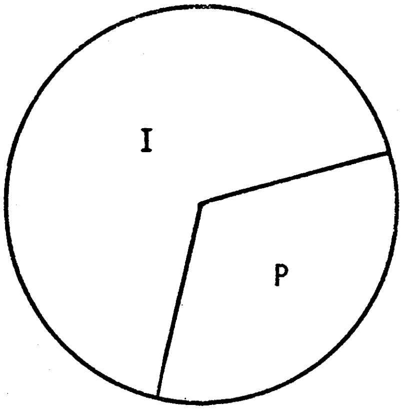
Description: Explanation of features of the UNIVAC 1108, Text Editor and its use at demand terminals, creating and modifying data or files.

Objectives: To gain ability to utilize more powerful and convenient methods of manipulating Source Files from conversational terminals.

Text Provided: Text Editor.

Pretest: None.

Time Distribution:



Instruction: Lecture.
Participation: Discussion.
Homework: None.

Schedule: Five 2-hour sessions; $2\frac{1}{2}$ weeks; two classes/year.

ADDITIONAL COVERAGE IN ADP COURSE CAPABILITY

1. Systems Analysis and Design Course: This is a 30-hour course of self-study, utilizing textual, audio, and video media (no single medium over 15 minutes at one specific period). It is an extensive, self-study program to develop skills in systems work for data processing personnel (DELTAK, Schiller Park, Illinois).
2. ADP Terminology: This is a course completely in audio, on audio tapes for personnel who wish to become familiar with data processing terminology. One of the recommendations for taking this course is to utilize a tape unit in a car going to and from work, listening and repeating the words, as well as the explanation following. This course was developed by a firm in Ohio, and is being utilized throughout the United States (DYNAPHONICS, Dayton, Ohio).
3. JPL/OS: The operating system for IBM's 360/75 computers. This program is completely on video tape for self-instruction or reinforcement, primarily to be used by the computer personnel having advanced knowledge of computer programs.
4. FORTRAN Fundamentals: In addition to the classroom presentation, this course can also be taken on a self-instructional basis by video tape.
5. Additional Courses: Additional courses are given in integrated circuit design, computer-aided circuit analysis, integrated circuit application, speed reading, telephone practice, computer concepts, and techniques, utilizing audio/video (animation) program, and workbooks on various subjects in the computer field in self-study mode, self-study audio cassette/workbook in BASIC, FORTRAN, and a computer technology course put out by the AIAA discussing computer technology media in the next decade.

APPENDIX B—COMPUTER-RELATED OCCUPATIONAL DESCRIPTIONS*©

OCCUPATIONAL DESCRIPTIONS

The occupations described on the following pages are considered to be the basic occupations most directly concerned with electronic computing. These occupations, however, are still fluid.

Hiring requirements and qualifications for employment have not yet been fully standardized. As a consequence, a certain amount of overlap and duplication of job duties and responsibilities will be noted among the various occupational descriptions. The occupational data, however, were assembled from many different sources and are a reflection of the existing situation. Since the data reflect the occupational situation as it exists in varied localities and establishments, the descriptions must be considered as composites of jobs, and cannot be expected to coincide exactly with any job in a specific organization. It will be necessary, therefore, to adapt descriptions to fit individual jobs before they can be used with complete accuracy.

The job descriptions are arranged in seniority order of their main titles. The wording of the title that appears at the head of each description is a reflection of common usage. Other titles, by which the job is known, also appear at the head of each description in small type. Between the main and alternate titles appears the code number which identifies the job within the present classification structure of the U.S. DICTIONARY OF OCCUPATIONAL TITLES.

The narrative portion of each job description is arranged as follows:

Occupational Definition—This provides a brief description of the duties involved in a particular occupa-

* © Edutronics Systems International, Inc., 1969 (Reprinted, with permission, from *Computer Education Directory*, Edutronics Systems International, Inc., Los Angeles, California, 1969.)

tion. It provides an understanding of the tasks that are performed, and the skills and knowledge that are necessary to the performance of those tasks.

Education, Training, and Experience—This section provides an indication of the amount of education and the level of training and experience usually required by management for employment in the occupation. As previously mentioned, the various occupations and the qualifications are not standardized, and considerable variation exists among employers as to required education, training, and experience. However, an attempt was made to indicate the range of such hiring requirements.

Special Characteristics—This section provides some estimate of the worker trait requirements pertinent to the specific occupations. It has long been believed that the ability of an individual to adjust to specific types of work situations is as significant as the education and training qualifications he brings to the occupation. This seems particularly significant when dealing with a group of relatively new occupations. Consequently, judgments have been in terms of a number of components consisting of aptitudes, interests, temperaments, physical activities, and environmental conditions to which individual workers have to adjust.

Aptitudes—These are the specific capacities or abilities required of an individual in order to facilitate the learning of some task or duty. This component is made up of 11 specific aptitude factors, and is a modification of the aptitudes contained in the General Aptitude Test Battery developed in the U.S. Employment Service. Those aptitudes were selected which seem to be significant in the occupation and were identified in terms of specific work situations. The factor of intelligence, however, was not rated because of the difficulty in writing meaningful descriptive statements.

Interests—This component is defined as a preference for a particular type of work experience. It consists of five pairs of bipolar factors, arranged so that a preference for one factor in a pair generally indicates a lack of interest in the other factor in the pair. Those factors were identified which seemed to be significant to the job in question, and were identified in terms of specific worker situations.

Temperaments—The temperament component consists of 12 factors defined in terms of specific work situations that suggest different temperament trait requirements. Each work situation describes a type of activity that demands a different adjustment on the part of individual workers. Those temperament factors were selected that appeared to be significant in the occupation, and were identified in terms of specific work duties.

Physical Activities and Environmental Conditions—This refers to (a) the physical activities required to be met by the worker; and (b) the physical surroundings which make specific demands upon a worker's physical capacities. There are six physical activities factors and seven environmental conditions factors. Those factors were selected that were significant in the occupation in the sense that they met established criteria for successful performance.

MANAGER, DATA PROCESSING—(169.168), DIRECTOR, DATA PROCESSING

Occupational definition

Directs and coordinates planning and production activities of electronic data processing division. Consults with management to define boundaries and priorities of tentative projects, discusses equipment acquisitions, determines specific information requirements of management, scientists, or engineers, and allocates operating time of computer systems. Confers with department heads involved with proposed projects to ensure cooperation and further defines nature of project. Consults with SYSTEMS ENGINEER, ELECTRONIC DATA PROCESSING to define equipment needs. Reviews project feasibility studies. Establishes work standards. Assigns, schedules, and reviews work. Interprets policies, purposes, and goals of organization to subordinates. Prepares progress reports to inform management of project development and deviation from predicted goals. Contracts with management specialists or technical personnel to solve problems. Revises computer operating schedule to introduce new program testing and operating runs. Reviews reports of computer and peripheral equipment production, malfunction, and maintenance to ascertain costs and plan operating changes within his department. Analyzes data requirements and flow to recommend reorganization or departmental realignment within the company. Participates in decisions concerning personnel staffing and promotions within electronic data processing departments. Directs training of subordinates. Prepares proposals and solicits purchases of analysis, programming, and computer services from outside firms.

Education, training, and experience

Two years of formal post-high school training in data processing with courses in business administration and accounting or engineering, or equivalent practical experience is necessary for small computer installations. College graduation with major in one or more of the above fields is preferred for large installations. Experience in systems analysis, programming, and computer operations is desirable. In installations offering more sophisticated services, such as operations research and engineering simulation, a college mathematics major coupled with experience listed above is desirable.

Special characteristics

Aptitudes:

Verbal ability to translate technical terminology into terms understandable to management and department heads.

Numerical ability to apply knowledge of linear or differential equations in evaluating work of department, preparing reports and proposals for management, and selling services to outside users.

Spatial ability to read engineering drawings, charts, and diagrams; to understand presentations, proposed solutions, and progress reports of business or engineering problems.

Form perception to see pertinent detail in drawing charts, diagrams, and other presentations.

Clerical perception to detect and avoid errors in reading and preparing reports.

Interests:

An interest in scientific and technical subjects to cope with wide range of technical and scientific problems processed through computer.

A preference for business contacts with people in directing activities of computer department and to sell ideas or services.

Temperaments:

Must be able to direct, control, and plan the operations of the division, bringing together knowledge of operations and information needs of departments, such as accounting, purchasing, engineering, sales, and inventory control.

Required to deal with people such as management, department heads, and manufacturers' representatives to exchange information and ideas, discuss equipment and their uses, elicit information from departments, and to answer inquiries from department heads.

Ability to influence management and department heads to enlist their support for acceptance, expansion, and sophistication of computer systems; and ability to sell excess computer time and services to outside clients.

Required to make judgmental decisions concerning equipment needs, scope of assignments, allocation of computer time, and organization of departments.

Required to make decisions based on factual data to evaluate progress or success of computerized projects.

Physical Activities and Environment:

Work is sedentary.

Occasionally walks to various departments and offices; stands during conferences and discussions with management, department heads, and supervisor of computer operations.

Talking and hearing required in conferences and during exchanges of information.

Near visual acuity and accommodation required for reading reports and charts.

Work is performed inside.

PROJECT DIRECTOR, BUSINESS DATA PROCESSING (020.168), BUSINESS-SYSTEMS COORDINATOR; LEAD ANALYST; PROGRAM MANAGER; PROJECT PLANNER; SENIOR SYSTEMS ANALYST, BUSINESS

Occupational definition

Plans, directs, and reviews business electronic data-processing projects, coordinates planning, testing, and operating phases to complete project with maximum balance of planning and equipment time, man hours and new equipment expenditures. Prepares project feasibility and progress reports. Confers with department heads who provide input and use output data to define content and format. Schedules and assigns duties to OPERATIONS RESEARCH ANALYSTS based on evaluation of their knowledge of specific disciplines. Coordinates activities of workers performing successive phases of problem analysis, solution outlining, solution detailing, program coding, testing, and debugging (error elimination). Reviews output data and related reports, applying knowledge of systems, procedures, methods, and data requirements of management and other output users to ensure adherence to predetermined standards and devise techniques for improved performance on similar future projects. Directs revision of continuous control project to adapt it to new data requirements or improve operations by using new techniques or equipment.

Education, training, and experience

A bachelor's or master's degree in business administration, with extensive course content in accounting and mathematics or statistics often is required. Employers frequently waive academic training requirements for currently employed workers with extensive systems analysis, design and followup responsibility in electronic data processing, and supervisory experience in tabulating-machine departments. Employers frequently require a degree or equivalent experience either in industrial engineering, or the engineering discipline most directly related to their manufacturing processes, when expanding business data processing to an integrated system that includes production forecasting, planning, and control.

Background knowledge and experience usually include a minimum of one to three years' experience in systems analysis and concurrent familiarization with structure, work flow requirements, and standards of the employing organization.

Current trend is toward greater mathematical sophistication than previously expected of workers at this level. The need for advanced mathematics becomes more urgent as computer applications become involved not only with normal business data-processing, but also with the involved problems of operations research.

Special characteristics

Aptitudes:

Verbal ability to elicit information and discuss project intentions, problems and progress, and to prepare reports.

Numerical ability to analyze problems and develop systems statements in form capable of being programmed. Mathematics varies from arithmetic and algebra for simple, single-purpose systems design to differential equations and mathematical statistics for complex systems involving optimization, simulation, or forecasting.

Spatial ability to develop, interpret, or integrate operational workflow diagrams and charts.

Clerical perception to recognize pertinent detail and avoid perceptual errors when working with verbal material, which often is in highly contracted and conventionalized form.

Interests:

A preference for prestige-type activities, and for business contact with others to participate in conferences with management, advise and inform others regarding the potentialities, limitations, and alternative methods of data processing, supervise analysts and coordinate their activities.

A preference for activities that are technical in nature to read and keep informed of computer development and, new or more refined systems and procedures techniques.

Temperaments:

Ability to perform a variety of duties involving frequent change, ranging from direct involvement in problem analysis to coordination of subsequent work processes until each project is operational.

Must be able to direct and plan an entire area of work activity, assigning subordinates on the basis of knowledge of individual specializations and abilities, and control activities through personal contact and reports.

Must be able to deal with people in nonsupervisory situations requiring considerable tact, to secure cooperation from management and all personnel affected by project.

Required to make decisions on a judgmental basis when supervising others and developing approaches to problems on basis of past experience. Evaluates fixed cost, time, manpower allocation, and output specifications to judge efficiency of project development and operation.

Required to make decisions based on factual data, as in planning proposed system around capabilities and limitations of a specific computer system.

Physical Activities and Environment:

Work is sedentary, with occasional standing and walking required.

Occasionally lifts and carries books, charts, diagrams, and other records seldom exceeding 10 pounds.

Talking and hearing to discuss problems and progress.

Near visual acuity to work with reports, charts, and diagrams and other printed or written records.

Work is performed inside.

SYSTEMS ANALYST, BUSINESS DATA PROCESSING—(012.168), COMMERCIAL-SYSTEMS ANALYST AND DESIGNER; DATA-METHODS ANALYST; SYSTEMS AND PROCEDURES ANALYST, BUSINESS DATA PROCESSING

Occupational definition

Analyzes business problems, such as development of integrated production, inventory control and cost analysis, to refine its formulation and convert it to programmable form for application to electronic data processing system. Confers with PROJECT DIRECTOR, BUSINESS DATA PROCESSING and department heads of units involved to ascertain specific output requirements, such as types of breakouts, degree of data summarization, and format for management reports. Confers with personnel of operating units to devise plans for obtaining and standardizing input data. Studies current, or develops new systems and procedures to devise workflow sequence. Analyzes alternative means of deriving input data to select most feasible and economical method. Develops process flow charts or diagrams in outlined and then in detailed form for programming, indicating external verification points, such as audit trial printouts. May work as member of team, applying specialized knowledge to one phase of project development. May coordinate activities of team members. May direct preparation of programs.

Education, training, and experience

College graduation with courses in business administration and accounting usually is required for entrants without prior experience in data processing. Some employers, while requiring a college degree, do not require a specific major or course content. A successful college record is regarded as proof of ability to reason logically which is considered more important for successful performance than knowledge of techniques acquired in any specific area. Many employers waive the formal education requirements for those workers employed in their establishments who have had several years' manual and machine systems experience prior to computer conversion. Business programmers without a college degree can, through experience, acquire a background in business systems and procedures and may thereby advance into systems analysis. Currently, the trend is to require a knowledge of advanced mathematics because of the rapidly increasing sophistication of business systems. Continuing education, through specialized courses, self-study, and participation in activities of professional associations, is the rule rather than the exception in this occupation, as in all higher level occupations related to the computer.

Special characteristics

Aptitudes:

Verbal ability to discuss problems and progress, prepare reports, and make annotations for graphic representations of work.

Numerical ability to select from alternatives to develop optimum system, procedures and methods. Mathematical investigation of such factors as variation in volume of input data, and frequency of appearance of exceptions to normal workflow in processing is often necessary. Level of mathematics varied from business arithmetic and algebra to differential equations.

Spatial ability to visualize, prepare, and review two-dimensional graphic representations of workflow.

Form perception to identify nonverbal symbols on records such as block diagrams and flow charts.

Clerical perception to avoid perceptual errors and recognize pertinent detail in the recording and identifying of letters and numbers that often occur in abbreviated or acronymic combinations.

Interests:

A preference for activities that are technical and analytical, and those that are abstract and creative in nature to devise new or to modify standardized computer-oriented systems to meet the specific needs of an organization.

Temperaments:

Ability to confer with personnel from other departments, develop flow charts, devise workflow sequence, and prepare reports.

Required to deal with people in conference and interview situations.

Required to make judgmental decisions to select from alternatives when devising optimal system.

Required to make decisions on basis of factual data to design system within machine capability.

Physical Activities and Environment:

Work is sedentary, with occasional standing and walking. Occasional handling of source documents, books, charts, and other records that seldom exceed 10 pounds.

Talking and hearing to discuss and confer with management and technical personnel.

Near visual acuity to prepare and review workflow charts and diagrams.

Work is performed inside.

OPERATIONS RESEARCH ANALYST—(020.088), MANAGEMENT-OPERATIONS ANALYST; OPERATIONS ANALYST

Occupational definition

Formulates mathematical model of management problems by application of advanced mathematics and research methods to provide quantitative basis for planning, forecasting, and making decisions. Analyzes problems in terms of management information requirements. Studies problem, such as selecting from competitive proposals a plan that affords maximum probability of profit or effectiveness in relation to cost or risk. Prepares mathematical model of problem area in form of one or several equations that relate constants and variables, restrictions, alternatives, conflicting objectives, and their numerical parameters. Gathers, relates, and identifies data with variables in model by applying personal judgment and mathematical tests. Specifies manipulative and computational methods to be applied to formulations and refers to data processing division for solving equations, programming, and processing. Reviews operations and testing of model to ensure adequacy or determine need for reformulation. Prepares written, nontechnical reports to management, indicating problem solution or range of possible alternatives in rank of desirability and probability of success when there is no single solution. Writes followup reports, evaluating effectiveness of research implementation. May specialize in research and preparation of contract proposals specifying the competence of an organization to perform research, development, or production work. May develop and apply time and cost networks, such as Program Evaluation and Review Technique (PERT), to plan and control large-scale business projects. May work in areas associated with engineering, as when analyzing and evaluating alternative physical systems, such as production processes, in terms of effectiveness and cost. May work alone or as member of a team.

Education, training, and experience

College degree with emphasis on advanced mathematics and statistics is usually the minimum educational requirement. A combination of advanced degrees in mathematics and business administration is especially desirable. A doctorate in mathematics is frequently required. Specific training in operations research at the graduate level is rapidly becoming a standardized requirement, as more schools offer courses in this interdisciplinary occupational area. Many workers have acquired the necessary background in mathematics through education and experience in engineering and the physical sciences, and knowledge of specialized techniques through self-study and participation in activities of professional organizations.

Special characteristics

Aptitudes:

Verbal ability to understand technical languages of various professional disciplines such as engineering and accounting, to give oral reports and to prepare written reports on results of research in lay terminology to management.

Numerical ability to understand and work with such mathematical specializations as game, queuing, and probability theory, and statistical inference to prepare formulations, specify manipulative methods and evaluate effectiveness.

Spatial ability to prepare and interpret charts, diagrams, graphs, and maps.

Clerical perception to recognize pertinent detail in compilation and analysis of statistical data, and to avoid perceptual errors in working with higher forms of mathematics.

Interests:

A preference for activities that are technical in nature to apply analytical, experimental, and quantitative techniques in the solution of management problems such as long-range forecasting, planning, and control.

Interests in devising mathematical equations, analyzing the methods used for their manipulation, and evaluating their practical effectiveness.

Temperaments:

Requires ability to perform a variety of tasks related to the solution of various problems on all departmental levels. This involves conversing in several professional disciplines with personnel at all operating levels to gather and relate data and opinions relevant to problem under study.

Must possess ability to make judgmental decisions such as probability of continuity or change in conditions, and assign arbitrary weights and values to problem factors when conventional statistical methods are not applicable.

Must possess ability to make decisions based on verifiable data, such as tabular records of previous organizational experience.

Physical Activities and Environment:

Work is sedentary, and occasionally involves lifting and carrying books, ledgers, and statistical tabulations seldom exceeding 10 pounds.

Talking and hearing to discuss organization goals and priorities with management and acquire data pertinent to the problem from other organizational personnel.

Near visual acuity to read and work with a variety of data from many sources, and to refer to texts and technical papers.

Work is performed inside.

CHIEF PROGRAMMER, BUSINESS—(020.168), COORDINATOR, COMPUTER PROGRAMMING; LEAD PROGRAMMER

Occupational definition

Plans, schedules, and directs preparation of programs to process business data by electronic data processing equipment. Consults with managerial and systems analysis personnel to clarify program intent, indicate problems, suggest changes and determine extent of automatic programming and coding techniques to use. Assigns, coordinates, and reviews work of programming personnel. Develops own programs and routines from workflow charts or diagrams. Consolidates segments of program into complete sequence of terms and symbols. Breaks down program and input data for successive computer passes, depending on such factors as computer storage capacity and speed, extent of peripheral equipment, and intended use of output data. Analyzes test runs on computer to correct or direct correction of coded program and input data. Revises or directs revision of existing programs to increase operating efficiency or adapt to new requirements. Compiles documentation of program development and subsequent revisions. Trains subordinates in programming and program coding. Prescribes standards of terminology and symbology to simplify interpretation of programs. Collaborates with computer manufacturers and other users to develop new programming methods. Prepares records and reports.

Education, training, and experience

Graduation from a technical school or college with training in business administration, computer programming, data processing mathematics, logic, and statistics is the usual educational requirement. Usually, a minimum of two years' experience in programming for same or similar computer system, on broadscope and complex projects is required. Experience should indicate knowledge of organization structure and workflow, and also reflect proven ability to supervise others and coordinate work activities of the group supervised with that of other organizational units.

Special characteristics

Aptitudes:

Verbal ability to present oral and written reports and recommendations and to read technical literature about changes in techniques and equipment.

Numerical ability to program at level of linear and Boolean algebra (logic) to minimize expensive programming and debugging time. Mathematics at level of differential equations and probability theory frequently required when an organization is developing or using sophisticated, integrated management information and forecasting systems.

Spatial ability to interpret systems statement, develop general and detailed computer flow charts, and prepare block diagrams that indicate hardware configuration.

Form perception to see pertinent detail in charts, diagrams, and code sheets composed of symbols.

Clerical perception to refer to manuals and written instructions and to review own work. This requires accurate identification of numbers, letters, words, and acronyms as well as ability to grasp general content.

Interests:

A preference for activities that are technical in nature to apply mathematics and logic in converting proposed business systems to computer-processable form.

Temperaments:

Ability to perform a variety of duties, covering problems from different but related areas of business activity such as production and inventory control, and sales analysis.

Must be able to direct, control, and plan development of business data processing programs that will meet current and future needs. This involves direction of activities, such as program testing and revising, and coordination of all phases of business programming activities to meet schedules.

Required to deal with subordinates for purposes of control and coordination, and with systems analysis and management personnel to resolve questions of intent and programming difficulties.

Required to make judgmental decisions based on experience to assign personnel and select and apply techniques that will produce least cost, fastest, or most flexible programs.

Required to make decisions based on factual data to evaluate adequacy of completed program by comparing program, test and operating runs with prescribed standards of terminology, and time.

Physical Activities and Environment:

Work is sedentary. Lifts and carries source and output data, books, charts, and diagrams seldom exceeding 10 pounds.

Stands, walks, talks, and hears in instructional and conference situations.

Near visual acuity to prepare, integrate, or modify complex programs, using a variety of charts, diagrams, and handbooks.

Work is performed inside.

PROGRAMMER, ENGINEERING AND SCIENTIFIC—(020.188), PROGRAMMER, TECHNICAL

Occupational definition

Converts scientific, engineering, and other technical problem formulations to format processable by computer. Resolves symbolic formulations, prepares logical flow charts and block diagrams, and encodes resolvent equations for processing by applying knowledge of advanced mathematics, such as differential equations and numerical analysis, and understanding of computer capabilities and limitations. Confers with engineering and other technical personnel to resolve problems of intent, inaccuracy, or feasibility of computer processing. Observes or operates computer during testing or processing runs to analyze and correct programming and coding errors. Reviews results of computer runs with interested technical personnel to determine necessity for modifications and rerun. Develops new subroutines for a specific area of application, or expands on applicability of current general programs, such as FORTRAN, to simplify statement, programming, or coding of future problems. May supervise other programming personnel. May specialize in single area of application, such as numerical control, to develop processors that permit programming for contour-controlled machine tools in source-oriented language.

Education, training, and experience

College degree with major in mathematics or engineering is usually the minimum educational requirement. A master's degree or doctorate in mathematics or engineering is a common requirement where analysis or programming is extremely complex or where work duties involve basic research in mathematics or programming. From two to four years of on-the-job training, with gradually decreasing amounts of supervision and with increasingly complex work assignments, are regarded as necessary for the worker to become familiar with at least one class of computer, programming language, and applications area. Short (one to four weeks) training sessions are given by employers and computer manufacturers to provide basic training and (later) specialized training. Current trends toward simplification of programming languages and greater applicability of generalized programs, and requirement of basic computer orientation and programming courses for college degree in physical sciences or engineering will reduce on-the-job training time.

Special characteristics

Aptitudes:

Verbal ability to discuss problems and equipment requirements with other technical personnel, to prepare computer flow charts, written records, reports, and recommendations and to read technical publications.

Numerical ability to interpret mathematical formulation, to select from alternative computational methods, to frame program within limitations of the computer to be used, prepare logical flow charts and diagrams, to convert program steps to coded computer instructions, and to review work. Level of mathematics may vary from arithmetic and algebra to advanced differential equations in the course of writing a single program, and may also include applications of numerical analysis.

Spatial ability to prepare logical flow charts specifying sequences of operating instructions and flow of data through computer system, to design input and output forms, and to interpret detailed drawings, diagrams, and other graphic data.

Form perception to see pertinent detail in charts, diagrams, and drawings, and distinguish symbols in subject matter areas such as physics and electrical engineering.

Clerical perception to avoid perceptual errors in recording of alphanumeric and special symbologies.

Motor coordination to operate calculator or computer.

Interests:

A preference for activities technical in nature to use mathematics to reduce formulations to computer-processable form.

Temperaments:

Required to make decisions on a judgmental basis, using past experience and knowledge to select best method of programming and coding a problem and thereby avoiding costly, time-consuming analyses of alternate techniques.

Required to make factual decisions, such as evaluation of program accuracy by computer acceptance, presence of error messages or obvious output distortions.

Must be able to conform to accepted standards and techniques in developing and testing programs, and in writing instructions for digital-computer operator.

Physical Activities and Environment:

Work is sedentary, requiring occasional lifting and carrying of source data, books, charts and diagrams seldom exceeding 10 pounds.

Talking and hearing to confer and collaborate with other technical personnel.

Near visual acuity to prepare logical flow charts from lengthy mathematical statements of problem, and to convert program steps to coded computer instructions.

Work is performed inside.

PROGRAMMER, BUSINESS—(020.188), DIGITAL-COMPUTER PROGRAMMER

Occupational definition

Converts symbolic statement of business problems to detailed logical flow charts for coding into computer language. Analyzes all or part of workflow chart or diagram representing business problem by applying knowledge of computer capabilities, subject matter, algebra, and symbolic logic to develop sequence of program steps. Confers with supervisor and representatives of departments concerned with program, to resolve questions of program intent, output requirements, input data acquisition, extent of automatic programming and coding use and modification, and inclusion of internal checks and controls. Writes detailed, logical flow chart in symbolic form to represent work order of data to be processed by the computer system, and describe input, output, arithmetic, and logical operations involved. Converts detailed, logical flow chart to language processable by computer (PROGRAMMER, DETAIL). Devises sample input data to provide test of program adequacy. Prepares block diagrams to specify equipment configuration. Observes or operates computer to test coded program, using actual or sample input data. Corrects program error by such methods as altering program steps and sequence. Prepares written instructions (run book) to guide operating personnel during production runs. Analyzes, reviews, and rewrites programs to increase operating efficiency or adapt to new requirements. Compiles documentation of program development and subsequent revisions. May specialize in writing programs for one make and type of computer.

Education, training, and experience

Minimum requirements are high school graduation with six months to two years of technical training in computer operations and in general principles of programming and coding, or equivalent job experience in these areas. Current trend is to hire college graduates for promotional potential with training in accounting, business administration, and mathematics, and provide them with a year of on-the-job training to qualify them for programming. In installations concerned with the application of the computer to more complex areas such as market research and statistical forecasting, a college degree in mathematics is preferred.

Special characteristics

Aptitudes:

Must possess verbal ability to understand and analyze oral or written statements concerning a variety of business problems and to discuss them with others.

Must possess numerical ability to interpret workflow charts, program problems, and understand machine logic. Level of mathematics varies from arithmetic and algebra for simple business data processing problems to differential equations and mathematical statistics for involved problems such as forecasting or optimization.

Must possess spatial ability to interpret diagrammatic representations of workflow, and to visualize flow of data through computer system to prepare computer block diagrams and logical flow charts.

Must possess form perception to see pertinent detail in symbols when reading, interpreting, or preparing charts, diagrams, and code sheets.

Clerical perception to detect errors in letters, words, and numbers recorded on charts, diagrams, and code sheets.

Interests:

An interest in activities technical in nature to effectively analyze problems, and to design logical flow charts and block diagrams.

An interest in activities that are carried out in relation to processes, techniques, and machines to plan sequence steps, to prepare instructions, and to test programs.

Temperaments:

Required to make judgmental decisions to plan logical sequence of steps and prepare logical flow chart for a project, keeping in mind capacities and limitations of computer and integrated machine units.

Must be able to conform to accepted standards and techniques in developing and testing programs, and writing instructions for computer operators to follow.

Physical Activities and Environment:

Work is sedentary, requiring occasional lifting and carrying of such items as source materials, run books, and documentations seldom exceeding 10 pounds.

Talking and hearing to communicate with systems, program coding, and operating personnel.

Near visual acuity and accommodation required to review statistical data and interpret charts and diagrams.

Work is performed inside.

PROGRAMMER, DETAIL—(219.388), JUNIOR PROGRAMMER; PROGRAM CODER

Occupational definition

Selects symbols from coding system peculiar to make or model of digital computer and applies them to successive steps of completed program for conversion to machine-processable instructions. Reads and interprets sequence of alphabetic, numeric, or special characters from handbook or memory for each program step to translate it into machine language or pseudo (symbolic) code that can be converted by computer processor into machine instructions. Records symbols on worksheet for transfer to punchcards or machine input tape. Marks code sheet to indicate relationship of code to program steps to simplify debugging of program. Confers with programming personnel to clarify intent of program steps. Usually works as understudy to PROGRAMMER, BUSINESS performing such additional tasks as converting flow charts and diagrams of simple problems from rough to finished form, or making minor changes in established programs to adapt them to new requirements.

Education, training, and experience

Must be high school graduate. Some training in programming, coding, and computer operations at technical school level is desirable. Experience in computer operations is preferred, but six months of job experience, most often in a clerical capacity, to become familiar with company operations, workflow, standards, and terminology is the minimum requirement. One to four weeks classroom training in coding for specific computer is usually provided by employer or computer manufacturer. Some employers favor college graduates in order to enhance the worker's promotion potential.

Special characteristics

Aptitudes:

Verbal ability to recognize programming and coding language consisting of abbreviated words, grouping of numbers, symbols, or mnemonics.

Numerical ability to convert decimal numbers to binary or other number systems.

Form perception to recognize and remember graphic symbols (arrangement of lines and curves).

Clerical perception to select from handbooks codes acceptable to computer, which involves accurate identification and recording of numbers, letters, and words.

Interests:

A preference for activities of a routine, concrete and organized manner to code programs, using standardized codes.

Temperaments:

Ability to perform repetitive tasks according to set procedures, to refer to source books for symbologies and select appropriate codes for each detailed step in program. Steps may number into the thousands.

Required to work to precise and established standards of accuracy in recognition of steps of detailed program and in selecting and recording codes. Any mistake can introduce error into programs and either prevent processing or distort output of program when processed.

Physical Activities and Environment:

Work is sedentary. Work sheets and code books seldome exceed five pounds.

Continuously handles pencils, work sheets, and code books while interpreting program steps, finding representative coding, and posting codes to work sheets.

Near visual acuity to select correct alphabetical, numerical, or special symbols to convert detailed program to machine-processable form.

Work is performed inside.

SUPERVISOR, COMPUTER OPERATIONS—(213.138), CHIEF CONSOLE OPERATOR; SENIOR CONSOLE OPERATOR; SUPERVISOR, DATA PROCESSING; SUPERVISOR, ELECTRONIC DATA PROCESSING

Occupational definition

Supervises and coordinates activities of workers who operate electronic data processing machines. Assigns personnel and schedules workflow to facilitate production. Directs training or trains personnel in operation of computers and peripheral and off-line auxiliary equipment. Works with programming personnel in testing new and revised programs. Develops operating methods to process data, such as devising wiring diagrams for peripheral equipment control panels, and making minor changes in canned (standardized) programs or routines to modify output content or format. Directs insertion of program instructions and input data into computer, and observes operations. Aids operators in locating and overcoming error conditions. Makes minor program and input data revisions through computer console to maintain operations. Notifies programming and maintenance personnel if unable to locate and correct cause of error or failure. Revises operating schedule to adjust for delays. Prepares or reviews records and reports of production, operating, and down-time. Recommends changes in programs, routines, and quality control standards. Consults with MANAGER, ELECTRONIC DATA PROCESSING about problems, such as including new program testing and operating runs in schedule and arranging for preventive maintenance time. Coordinates flow of work between shifts to assure continuity. May supervise personnel engaged in keypunching, data typing, and tabulating.

Education, training, and experience

High school graduation is the minimum requirement. Usually a minimum of one to three years' experience in operating computers, peripheral, and off-line equipment is required. A familiarization with programming and coding techniques usually gained through experience in computer operation or a course in programming are additional prerequisites. However, a two-year post-high school training course in electronic data processing may reduce experience requirements. Training in business administration, mathematics, and accounting is regarded as particularly desirable. Some employers require a college degree, particularly in large installations where work duties are largely administrative, and also in order to enhance the worker's promotion potential. Experience in supervising personnel is desirable.

Special characteristics

Aptitudes:

Verbal ability to train and supervise subordinates, confer with other supervisory and technical personnel, and prepare records and oral or written reports.

Numerical ability to level of arithmetic and algebra to cope with error or stoppage situations in computer operations; to plan operating changes; and to prepare variety of reports, often on a daily basis such as revision of average time requirements for processing data or cost allocations to departmental users.

Spatial and form perception to prepare wiring diagrams, wire control panels for peripheral machines, and operate equipment.

Clerical perception to review records of production, operating and down-time, and recognize pertinent detail in computer printout dumps.

Interests:

A preference for business contact with others to confer with management, technical personnel, suppliers of input data, and users of computer output, and to supervise and train subordinates.

A preference for activities that are technical in nature to keep informed of new techniques for operation of current machines.

Temperaments:

Ability to perform a variety of tasks subject to frequent change, such as training and assigning personnel, accommodating changes in production schedules to meet priority runs, maintaining records, and developing new operating procedures.

Must be able to direct, control, and plan the activities of computer operators, satellite input-output computer operators, on-line and off-line peripheral equipment operators, and tape librarians.

Required to deal with people such as other departmental supervisors to resolve problems concerned with the scheduling, adequacy, and accuracy of input data.

Required to make decisions on a judgmental basis to set up work assignments that make maximum use of workers' knowledge and ability, and most effective and economical use of computers and peripheral equipment.

Required to make decisions on a factual basis when developing schedules for processing programs, relating such factors as date of program receipt, priority assignment, estimated run time, and available computer time. Compares output requirements against available equipment, and existing programs, routines, wiring diagrams, and control panels to determine need for developing or modifying operational methods, or altering operating schedule.

Physical Activities and Environment:

Work is light, requiring frequent standing and walking and occasional lifting and handling of reels of tape, decks of punchcards, and control panels weighing up to 20 pounds.

Talking and hearing to give oral instructions, assign work, and train personnel. Confers with management and others, discussing such items as budget requirements and staffing, machine capability, and production problems.

Near visual acuity to frequently analyze records, prepare reports, study program run books, and read technical literature.

Work is performed inside.

COMPUTER OPERATOR—(213.382), COMPUTER OPERATOR; CONSOLE OPERATOR

Occupational definition

Monitors and controls electronic digital computer to process business, scientific, engineering, or other data, according to operating instructions. Sets control switches on computer and peripheral equipment, such as external memory, data communicating, synchronizing, input, and output recording or display devices, to integrate and operate equipment according to program, routines, subroutines, and data requirements specified in written operating instructions. Selects and loads input and output units with materials, such as tapes or punchcards and printout forms, for operating runs, or oversees operators of peripheral equipment who perform these functions. Moves switches to clear system and start operation of equipment. Observes machines and control panel on computer console for error lights, verification printouts and error messages, and machine stoppage or faulty output. Types alternate commands into computer console, according to predetermined instructions, to correct error or failure and resume operations. Notifies supervisor of errors or equipment stoppage. Clears unit at end of operating run and reviews schedule to determine next assignment. Records operating and down-time. Wires control panels of peripheral equipment. May control computer to provide input or output service for another computer under instructions from operator of that unit.

Education, training, and experience

A high school education meets the minimum educational requirements of some employers, but an increasing number of employers are demanding an additional several months to two years of technical school training in data processing. This training usually includes such courses as data processing mathematics, accounting, business practices, elementary programming, and operation of computers, peripheral equipment, and tabulating machines.

The employer or computer manufacturer usually provides one to three weeks of formal instruction for the specific computer system the worker will operate. Length of subsequent on-the-job training and experience required to achieve adequate performance ranges from a few months to one year because computer systems and equipment vary in complexity and need for operator intervention. Except in small units, a minimum of three to six months prior experience in operation of peripheral equipment frequently is required.

Special characteristics

Aptitudes:

Verbal ability to comprehend technical language of operating instructions and equipment manuals and to explain clearly any operating problems and difficulties in interpreting program intent.

Numerical ability at level of arithmetic to prepare operating records, time computer runs, and adhere to fixed operating schedule. While not always an operator requirement, an understanding of data processing mathematics (the number systems used, algebra and logic) is almost essential to discuss operating difficulties with programming personnel, and to progress from routine production runs to the testing of new programs.

Spatial perception to wire control panels for peripheral equipment.

Form perception to identify flaws in input and output materials.

Clerical perception to avoid perceptual errors in preparing operating records, and to recognize alphabetic, numeric, and mnemonic symbols.

Motor coordination, to rapidly set up machines and to move keys and switches to quickly correct errors or stoppages.

Interests:

A preference for working with machines and processes to continuously operate and monitor the equipment that comprises the computer system.

Interest in activities of concrete and organized nature to operate machines according to specific and detailed instructions.

Temperaments:

Work situation requires ability to perform a variety of work tasks subject to frequent change in the simultaneous operation of a console and a variety of peripheral equipment, the integration of which varies from program to program, or even during a single operating run, and which demands rapid transfer of attention from one piece of equipment to another.

Accuracy required to operate system effectively and minimize down-time and rescheduling of runs. Carelessness in following written and oral instructions can cause extensive rebuilding of program or input data, or even lead to irrecoverable loss of data.

Physical Activities and Environment:

Work is light. Lifts, carries, and positions tape reels, punchcard decks, output forms, and control panels seldom exceeding 20 pounds.

Stands and walks frequently when loading and monitoring machines.

Reaches for and fingers switches and keys on console and peripheral machines. Wires control panels, loads and removes input and output materials.

Talking and hearing to frequently exchange information concerning program and system requirements with other workers and to receive or give instructions.

Near visual acuity and accommodation to follow detailed operating log, monitor computer and peripheral machines for signs of malfunction, and analyze console messages or high-speed printer output for cause of error or stoppage.

Color vision to distinguish between colored wires when wiring control panels, to identify color-coded cards or forms and to monitor colored display lights if used.

Work is performed inside.

COMPUTER-PERIPHERAL-EQUIPMENT OPERATOR—(213.382), ASSISTANT CONSOLE OPERATOR; TAPE HANDLER

Occupational definition

Operates on-line or off-line peripheral machines, according to instructions, to transfer data from one form to another, print output, and read data into and out of digital computer. Mounts and positions materials, such as reels of magnetic or paper tape onto spindles, decks of cards in hoppers, bank checks in magnetic ink reader-sorter, notices in optical scanner, or output forms and carriage tape in printing devices. Sets guides, keys, and switches according to oral instructions or run book to prepare equipment for operation. Selects specified wired control panels or wires panels according to diagrams and inserts them into machines. Presses switches to start off-line machines, such as card-tape converters, or to interconnect on-line equipment, such as tape or card computer input and output devices, and high-speed printer or other output recorder. Observes materials for creases, tears, or printing defects and watches machines and error lights to detect machine malfunction. Removes faulty materials and notifies supervisor of machine stoppage or error. Unloads and labels card or tape input and output and places them in storage or routes them to library. Separates and sorts printed output forms, using decollator, to prepare them for distribution. May operate tabulating machines, such as sorters and collators.

Education, training, and experience

High school graduate. Post-high school training in operation of electronic or electromechanical data processing equipment is desirable. Employers frequently regard worker as understudy to computer operators and apply same education and aptitude standards to them.

Special characteristics

Aptitudes:

Verbal ability to read written instructions and handbooks and to communicate with supervisor about operating functions.

Spatial ability to follow diagrams to wire control panels, position and thread tapes onto spindles or position decks of cards in hoppers.

Clerical perception to identify and record, without error, data such as dates, program numbers, departments, and routings on forms.

Motor coordination and finger and manual dexterity to load and unload machines quickly and minimize down-time, to thread ribbons of tape over guides and through rollers, and to handle cards and tapes deftly without bending, tearing, or otherwise damaging them.

Interests:

An interest in activities concerned with machines, processes, and techniques to operate various machines. Preference for activities of a routine and organized nature to follow well-defined instructions for any of several different machines.

Temperaments:

Worker must be adept at performing a variety of tasks requiring frequent change to operate a number of machines in varying combinations and sequences.

When operating peripheral equipment, must adhere to established standards for accuracy, such as observing printer output forms for defects in alinement, spacing, margin, and overprinting. Immediate response to indication of error in operation of peripheral equipment is vital.

Physical Activities and Environment:

Work is light, involving frequent standing and walking when operating machines and lifting and carrying tapes, cards, and forms not exceeding 20 pounds.

Reaches, handles, and fingers to mount tapes onto spindles, position decks of cards in hoppers, and thread tape through guides and rollers of peripheral units or wire control panels.

Near visual acuity to read labels on reels, to wire plug boards from diagrams, to scan printout for error, and to read operating instructions and handbooks.

Color vision to distinguish between various colors of wires to ensure correct wiring of control panels.

Work is performed inside.

CODING CLERK—(219.388)

Occupational definition

Converts routine items of information obtained from records and reports into codes for processing by datatyping or key-punch units, using predetermined coding systems. Manually records alphabetic, alphanumeric, or numeric codes in prescribed sequence on worksheet or margin of source document for transfer to punchcards or machine input tape. May be designated according to trade name of computer system as CODING CLERK, UNIVAC; IBM CODER.

Education, training, and experience

High school graduation usually is required. Training of a day or two in a classroom situation or under the direction of an experienced worker usually is provided by the employer. Achievement of adequate speed, and particularly the development of a high degree of accuracy, takes from one to three months. Achievement of speed involves memorization of many of the codes. Familiarization with standard business terminology and abbreviations, and with special conventions used by the employer can reduce the time necessary to achieve adequate performance.

Special characteristics

Aptitudes:

Verbal ability to understand written and oral instructions, business terms, abbreviations, and mnemonic contractions, and to explain difficulties to supervisor.

Clerical perception to scan and extract pertinent detail from source documents, and to avoid and detect perceptual errors.

Interests:

A preference for activities of a routine, concrete, organized nature to code according to a predetermined, standardized system.

Temperaments:

Work activities are repetitive and short cycle in nature. Converts each item into the equivalent code in the allowable time cycle of a few seconds.

Must follow specific instructions to convert items to their coded equivalents which are indexed in tables or handbooks.

Physical Activities and Environment:

Work is sedentary. Reaches for, lifts, and handles handbooks, papers, and forms seldom exceeding five pounds.

Near visual acuity to read and convert items to codes.

Work is performed inside.

TAPE LIBRARIAN—(223.387)

Occupational definition

Classifies, catalogs, and maintains library of magnetic or punched paper tape or decks of magnetic cards or punchcards used for electronic data processing purposes. Classifies and catalogs material according to content purpose of program, routine or subroutine, and date on which generated. Assigns code conforming with standardized system. Prepares index cards for file reference. Stores materials and records according to classification and catalog number. Issues materials and maintains charge-out records. Inspects returned tapes or cards and notifies supervisor if worn or damaged. May maintain files of program developmental records and operating instructions (run books). May operate key-punch to replace defective punchcards and produce data cards to identify punchcard decks. May work in computer room performing such tasks as loading and removing printout forms, reels of tape, and decks of cards from machines.

Education, training, and experience

High school graduate, preferably with commercial background. Three to six months experience as catalog clerk, file clerk, mail clerk, or messenger with the company is desirable.

Special characteristics

Aptitudes:

Verbal ability to read information on labels describing contents of decks of cards and reels of tape and read catalogs that contain standardized codes and abbreviations.

Numerical ability to count, add, and subtract numbers to perform inventory functions.

Clerical perception to note pertinent detail on labels, cards, or work schedules, and in code books to detect error and avoid misinterpretation.

Interests:

A preference for working with things and objects, following routine and organized patterns to classify, catalog, store, and fill requests for cards and tapes.

Temperaments:

Follows specific instructions and established procedures to receive, store, issue, and purge materials.

Physical Activities and Environment:

Work is light, involving frequent standing and walking to carry reels of tapes or trays and drawers of cards weighing less than 20 pounds between desk, handtruck, file cabinets, and racks.

Reaches for and handles tapes and cards to store them.

Near visual acuity to read and assign identification symbols and inspect materials for damage.

Work is performed inside.

KEY-PUNCH OPERATOR—(213.582), CARD-PUNCH OPERATOR; PRINTING-CARD-PUNCH OPERATOR; PRINTING-PUNCH OPERATOR

Occupational definition

Operates alphabetic and numeric key-punch machine, similar in operation to electric typewriter, to transcribe data from source material onto punchcards and to reproduce prepunched data. Attaches skip bar to machine and previously punched program card around machine drum to control duplication and spacing of constant data. Loads machine with decks of punchcards. Moves switches and depresses keys to select automatic or manual duplication and spacing, selects alphabetic or numeric punching, and transfers cards through machine stations. Depresses keys to transcribe new data in prescribed sequence from source material into perforations on card. Inserts previously punched card into card gage to verify registration of punches. Observes machine to detect faulty feeding, positioning, ejecting, duplicating, skipping, punching, or other mechanical malfunctions and notifies supervisor. Removes jammed cards, using prying knife. May tend machines that automatically sort, merge, or match punchcards into specified groups. May keypunch numerical data only and be designated KEY-PUNCH OPERATOR, NUMERIC.

Education, training, and experience

High school graduate preferred with demonstrated proficiency in typing on standard or electric typewriter. High school or business school training in key-punch operation is desirable. Frequently, one week of training is provided by employer or manufacturer of equipment.

Special characteristics

Aptitudes:

Verbal ability to understand oral and written instructions, such as manufacturers' operating manuals, and to learn operation of machine.

Clerical perception to perceive pertinent detail in tabular material consisting of combinations of letters and numbers, and avoid perceptual error in transferring this data to punchcards.

Motor coordination to read work sheets and simultaneously operate keyboard of approximately 40 keys to punch data on cards.

Finger dexterity to move switches on machine.

Interests:

Preference for organized and routine activities to transfer data onto punchcards.

Temperaments:

Must be able to perform repetitive duties of operating key-punch machine.

Ability to follow specific instructions and set procedures to transfer data onto punchcards.

Required to work to precise and established standards of accuracy to key-punch data at high rate of speed.

Physical Activities and Environment:
Work is sedentary with infrequent lifting of decks of cards when loading machine.

Reaches for and handles code sheets, business records, and decks of cards; fingers switches and keys to operate machine.

Near visual acuity to read copy when key-punching.

Work is performed inside.

The future of minicomputer programming

by D. J. WAKS and A. B. KRONENBERG

Applied Data Research, Inc.
Princeton, New Jersey

> *"Here is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it. And then he feels that perhaps there isn't."*
>
> Beginning of *Winnie-the-Pooh*, A. A. Milne

INTRODUCTION

The minicomputer syndrome

The programming of minicomputers to date has resembled Pooh Bear coming downstairs because although we feel that there really is another way, we rarely stop bumping. Programmers for small computers have long exhibited what we call "the minicomputer syndrome", which is a very low expectation of program support and facilities on a minicomputer and a lack of appreciation of the hardware and software tools one can employ to attack and solve a programming problem.

Minicomputers came on the market when it was first realized that small general-purpose stored-program computers could be competitive with complex hard-wired systems built from general-purpose logic modules. Not at all surprisingly, the manufacturers of logic modules quickly became the leading developers of minicomputers. Since the programming process was viewed as an unfortunately necessary step in making the mini behave like the hard-wired system it was replacing, very little general-purpose programming support was provided with the minis. Thus, the minicomputer syndrome—the belief that primitive computers require primitive programming support—was born virtually at the same time as the mini itself.

Computer manufacturers have fostered this syndrome by typically providing little general-purpose software* with their minis. Although manufacturers have recently attempted to provide reasonable program-creation software, the majority of available assemblers still lack such desirable features as macros, conditional assembly, literals, and multiple location counters. Program-execution software for most small computers is often so weak that the user usually is left to write his own disk handler, communications package, or real-time executive. Even in cases where a manufacturer does provide such software, it usually requires a configuration that is uneconomically large.

The manufacturers' software planners seem to exhibit the minicomputer syndrome more than many of their customers. They continue to design separate, single-application monitors apparently assuming that the user will employ his mini only for real-time applications, file processing, communications, or report writing. Thus, the user who wants a single monitor system to support some combination or all of these must either adapt a manufacturer-supplied monitor or build one from scratch.

This lack of coherent manufacturer software support has also been characterized by a lack of standardization and conventions or standardization of poor or unworkable conventions in the software. This in turn has led to a proliferation of software to dismay any ecologist. For example, how many different assemblers now exist for the DEC PDP-8 family? We were able to count a baker's dozen in two minutes of trying; there must be an equal number we didn't think of or don't know about.

Finally, this lack of standards and conventions has resulted in incompatibility from one user to another. Software modules created for one user's executive and assembler can be neither assembled nor run at another user's site. We have been involved in more than one

* General-purpose software can be classified as either program-creation software (text/file editors, language processors), or program-execution software (monitors, device drivers, real-time executives).

project requiring that a program be manually transliterated from one dialect for a computer to another.

What is a minicomputer?

The paper is intended to describe programming for minicomputers. What do *we* mean by "minicomputer"?

Let us begin by characterizing the *small computer* as having:

(a) A short word and register length, aiways 18 bits or less; a small computer, therefore cannot perform *single-precision* floating-point arithmetic through either the hardware or software.
(b) A main memory provided in modules of 8K words or less (typically 4K words).
(c) An instruction set and, commonly, its addressing mode which are restricted, usually having a rather short operation code field in its machine-language structure.

One more characteristic is required to define what we mean by a mini: if a machine passes the test for a small computer, and you can't ordinarily buy it without an ASR Teletype®, it is a *mini*. If you can't ordinarily buy it without a *card reader*, it *isn't* a mini, it's just a small computer.

The virtual machine and the extensible machine

Let us digress for a moment and define some useful concepts we will refer to throughout the remainder of this paper. The concept of the "extended machine" was first propounded by Holt and Turanski;[1] we will use the term "virtual machine" interchangeably with extended machine. Here we are using the term "virtual" in its dictionary[2] sense: "Virtual, apparent, as contrasted with actual or absolute." That is, the virtual or extended machine is what the computer user, on some level, sees and uses. Another way of saying this is to describe it as the *set of primitives* with which he must work. Watson[3] defines a "virtual processor" in a timesharing system in a similar way. The "virtual memory" concept in paged computers is a similar derivation.

As it is, the computer user rarely sees or uses the "actual or absolute" machine. To do so, he would have to write all his code in machine language. Language processors, executive systems, device drivers, *et al.*, are all extensions to the machine which substantially affect the user's view of the machine. For instance, a minicomputer with a disk operating system appears to be a very different machine from one with only paper-tape-oriented software. A mini with a manufacturer-provided real-time executive appears to be a very different machine from one without it to the user with a data acquisition and reduction problem. In this sense, then, all available software and hardware options act as real extensions to the "actual" machine.

When certain types of extensions are preplanned into the computer hardware and software design, the resultant machine is considered "extensible"—to the extent that such extensions are planned. We characterize these extensions as being expansions of primitives on three distinct levels: the hardware or "firmware" level; the system software level; and the applications software level. On each level, the user can implement new extensions, typically visible at the next higher level (further away from the hardware) although sometimes available at the same level.

The *first* and *lowest level* of extensibility is at the *hardware* or *firmware* level. By hardware extensibility, we mean the ability to add new operation codes, typically for I/O but also for CPU purposes. Thus, a *supervisor call* facility was added to a PDP-8 four years ago to provide for user-supervisor communication in a real-time system for TV control room automation. This form of extensibility is designed-in, and encouraged, although only rarely used to augment CPU primitives. By firmware extensibility, we mean extensions through modifications or additions to microcode on those computers (such as all machines in the Interdata family) which have a two-level (micro and user) programming structure.[6] This ability to modify or augment the instruction repertoire of the machine is a powerful way, albeit rarely exploited, to extend the machine at the lowest level; the new primitives thus created are fully comparable to the original instructions of the actual machine.

The *second level* of extensibility is at the software level through creation of *system software*. By system software, we mean such obvious extensions as operating systems, device drivers and interrupt handlers, and any program-creation software, especially language processors, which can drastically change the primitives available to the end user. *Planned* extensibility at this level implies such hardware features as:

(a) A user mode/supervisor mode distinction, providing a trap of some kind against the execution of reserved instructions in user mode, and providing some kind of "supervisor call" facility.
(b) An approach to I/O on the hardware level which

® Teletype, a registered trademark of the Teletype Corporation.

provides for substantial compatibility of the I/O mechanisms from device to device, particularly with respect to interrupts and status information.

(c) Some method for extended address mapping which provides for hardware protection and relocation, thus relieving the mini user of his worst problem—severe limitations on addressing.

The *third* and *highest level* of extensibility is on the user software level. Planned extensibility provides the end user with the capability of augmenting the apparent set of primitives he uses in writing his programs. Such long-used techniques as subroutines and macros both constitute expansions of primitives, particularly when used together, e.g., a macro used to generate a calling sequence for a subroutine which performs some complex function.

Many sophisticated programmers for minis, particularly those coming from a large-computer background, habitually define large numbers of macros to provide extended primitives comparable to those found in the actual instruction set on large machines. In this way, the experienced programmer of a minicomputer views the problem of programming a mini as being no different from that for a large computer, effectively countering the purveyors of the "minicomputer syndrome" by using the same techniques, approaches, and skills developed for large computers. The manufacturer can plan for extensibility on this level by providing the user with macro features in the assembler and by providing flexible hardware and software techniques for invoking subroutines.

Let us close this introduction by noting that each layer of extension, although providing a new set of primitives to the next higher level, may also reduce the original set of primitives available on that level. A set of device drivers for I/O handling, once defined, usually prevents the user from writing code to use a new I/O device in a substantially different way from that standardized by the drivers added as extensions; he also loses, in the process, any capability of doing I/O operations himself, since attempts to do so are trapped by the hardware.

Additionally, a Basic system—editor, compiler, and operating system combined—gives the Basic user a different and more powerful set of primitives than he has in assembly language, but deprives him of the ability to perform operations in assembly language, even though this might be far preferable for some aspects of a given problem. Thus, the extended machine may look to the user quite different from the original machine, with some primitives added and others deleted, at the extender's discretion.

The remainder of this paper is devoted to describing many kinds of extensions which we see occurring now and in the future. Since this paper is principally about software, we will concentrate on extensions to the second and third levels. However, we feel that current hardware developments will encourage software-like extensions at even the lowest level, and we will discuss these implications also. Hopefully, viewing the minicomputer as more closely related to the large-scale computer than the hard-wired controllers it was originally designed to replace will lead to alleviating the underlying symptoms of the minicomputer syndrome.

THE MINI AS A GENERAL-PURPOSE COMPUTER

We will first discuss the minicomputer as it may be viewed by the applications programmer in the future. To do this, we will first examine a significant difference in the evolution of larger computer software as opposed to minicomputer software.

Both scales of computers developed from the same base: machine-language programs loaded into memory via manually produced media (cards or paper tape) followed by console debugging with the user setting switches and reading lights on the computer. In the history of large computers, the first executive systems were batch monitors, which were quite widely developed by 1960. These batch monitors provided for submitting "jobs" on decks of cards to the computer center where they were eventually entered into the "job stream" and run. All programming and debugging tools developed for large computers during most of the 1960s were oriented to batch operation; and today virtually all operating systems for today's large computers are optimized for batch processing. Nearly all commercial programming, and most university programming, is today done using batch monitors on computers including the IBM 360 and 370, the Univac 1108, and the CDC 6600 and 7600. Recently, a trend has started toward providing some sort of support for interactive programming, debugging, and program execution. Except for several large computers explicitly designed for interactive operation in time-shared applications (specifically the XDS 940 and the DECsystem-10, both produced by companies previously known for minis), the machines were originally designed for batch operation and modified in both hardware and software to support interactive operation. The IBM 360/67 is a modification to the 360/65, the Honeywell (GE) 645 to the 635, the Spectra 70/46 to the 70/45. These machines seem to have been grudgingly provided by the manufacturers to meet what they saw as a small, uneconomical, but prestigious market; none of them provide nearly as good interactive operation as machines and software designed to be interactive from the beginning. In particular, a system like the DECsystem-10 can provide program interaction with the user on a character-by-character basis through the user's teletypewriter; no machine based on a 2741-type terminal can possibly do so, since the terminal and its supporting computer-based interfaces are designed to operate strictly on a half-duplex, line-at-a-time basis. But the trend would appear to be toward expanded interactive and conversational uses of large computers, particularly as management information systems make terminal access to data bases desirable in real time.

Minicomputer programming started in the same way as the large computers. Paper tape was the primary I/O medium, read in by an ASR-33 Teletype®, with the computer console used for all debugging. Since minicomputers cost so much less than large computers, there was much less pressure to get away from console debugging, particularly as it was recognized as being a very cost-effective tool for a competent programmer. When it became clear that it was possible to create software tools to facilitate debugging, it was natural to use the Teletype®, which was there anyway (recalling our definition of a minicomputer), as the I/O device around which to design the "monitor." The first interactive programs for minicomputers, using the "console teletypewriter" to provide interaction with the user, were symbolic editors and debugging tools (particularly DDT,[7] a debugging system produced at MIT for the PDP-1). Many other interactive systems for minis are described in an earlier paper by one of the authors.[8] Interactive systems of all kinds are common today on virtually all minicomputers. In addition to symbolic editors and debugging .systems, there are full-scale Basic systems for one or several users on many minis; several minis have Disk Operating Systems designed for conversational control from the console teletypewriter, single-user and multi-user interpretive systems derived from JOSS® running on one or more terminals, etc.

Now that minicomputer configurations often include peripherals, such as mass storage on removable disk packs, card readers, line printers, magnetic tapes, formerly found only on large computers, we see a trend toward the use of batch operating systems on minicomputers. Such systems, typically based on RPG and/or Fortran as higher-level languages, are closely modeled after large-machine batch systems. *The resulting virtual machine thus looks almost, if not completely, like the virtual machines erected on larger-machine bases.* A user programming in Fortran or RPG, both relatively standardized languages, cannot tell, when he submits a deck and gets back his reports, whether it was run on a large computer or on a mini (except, perhaps, by the size of the bill!).

Thus, large computers and minis are becoming more and more alike—at least from the point of view of the applications programmer. The authors hope that the significant advantages of interaction between the user and the computer, so prominent in the development of minicomputer software, will not be disregarded by the mini manufacturers in their seemingly headlong rush to "me-too" compatibility with larger computers and their batch operating systems.

THE MINICOMPUTER EXTENSION SPECIALIST

Until now, the people who have played the largest role in extending minicomputers have been the system software designers and implementors and the hardware engineers. We feel that the systems software designers will have an increasing role as extension specialists in the future by becoming knowledgeable in hardware techniques.

Hardware/software extensibility

Already, the manufacturers of minis are providing richer instruction sets and more designed-in extensibility in their newer computers. As an example, the DEC PDP-11 provides a rich instruction set, a novel I/O structure, and virtually a complete elimination of the addressing problem which once plagued most minis. The Interdata Systems 70 and 80 provide substantial encouragement for the user to employ microprogramming to extend the base machine, i.e., to produce firmware. Almost all new minis provide modular packaging, in which a hardware function, to add a new primitive, can be easily wired on a single module and plugged into the machine.

We see the continued growth of microprogrammed processors, built around Read-Only Memories (ROM's), as being in the vanguard of user-extensible machines. The newest hardware facilities include Programmable ROM's (PROM's) which can be written by the user on his site; Erasable PROM's—which can be manually

® JOSS, a registered trademark of the RAND Corporation.

erased after being written; and Writable ROM's (slow write cycle compared to read, intended to be written relatively infrequently), sometimes called "Read-Mostly Memories" (or RMM's). All of these lead us to see a trend toward making this form of extensibility available by design, rather than by accident.

Thus, the extension specialist will be encouraged to work on all levels of extensibility. He will be able to add firmware in the form of lower level programming (microprogramming) or as pre-wired primitives which he can literally plug in as new functions (as on the GRI-909 family). Writable ROM's promise additional extensibility, even providing the systems programmer with the ability to optimize the instruction set for each application by loading a writable ROM from his system initialization code. Thus, he might choose an instruction set built around four-bit decimal numbers to run Cobol or RPG programs, one featuring binary floating point on multiple-precision numbers for Fortran or Algol, one with extensive byte manipulation for Snobol, and one with primitive expression evaluation for PL/I.

Systems programmers will become more competent at lower-level extension work, either by the firmware being brought closer to the user in the form of writable ROM's, or by the software designer being cross-trained in hardware techniques. For many minicomputer systems programmers, an oscilloscope is a familiar tool in debugging complicated programs and systems. Every sign indicates a growing encouragement for the systems programmer to create his own desired environment through extensibility. The DEC PDP-16 carries this to its logical extreme by letting the user build his own computer from a pre-determined set of modules using software tools to aid the design and implementation process.

Thus, we predict that the systems designer of the future will increasingly be at home in both cultures—hardware and software.

Programming automation systems

We also see a substantial growth and extension to what we call "programming automation systems"—techniques that provide the programmer with computer aids in the programming process.[9] All programming systems are designed to help the programmer through the critical loop of program development, i.e., program modification, language processing, debugging, and back to modification. Thus, systems such as Basic and JOSS® provide for complete control over the critical loop within the context of a single system with a single language used to control all three functions. On minicomputers, symbolic editors, language processors and debugging aids are provided to "automate" these three steps. For the systems programmer working on logically complex and large programs, however, the minicomputer does not really provide an optimum environment. Unless it provides bulk storage and a line printer, it is not well suited to editing and assembly. Unless it has a console much better designed than most mini consoles, it is also not particularly well suited to debugging (and the newest consoles are even less useful for debugging). Thus, there seems to be a trend toward the use of larger computers, particularly time-shared systems, for supporting programming automation systems designed to support smaller machines. A large-machine-based editor and assembler can do a lot to facilitate the creation of programs, particularly since the assembler does not have to be restricted in either the features or the number of symbols which can be accommodated. A good simulator, designed for debugging, can significantly improve the productivity of a programmer by providing him with the necessary tools to debug a program without the limitations of the smaller machine. The authors have invested quite some time over the last few years investigating this approach;[10] several other organizations have also done so, with some success. Several new computers, notably the GRI-909 and the IBM System 7, accentuate this trend; they are only supported by larger host computers; the actual machines are configured only to run programs, not to create them.

For years, systems programmers for small machines have felt, like Pooh Bear, that there must be a better way than assembly language to write programs for small machines. The so-called "implementation languages" have been in use for some time on larger computers; languages such as AED,[11] BLISS,[12] PL/I and ESPOL[13] have been used to create systems software, including compilers and operating systems, for large computers. We regard it as unlikely that implementation languages will soon be operational on minicomputers, due to their high initial cost and inefficiency of generated code. (Only if substantial computer time is invested in optimizing time and space will minis be able to support such implementation languages.) We do feel that the trend toward using larger computers to support minis will continue, and that it will soon be possible for systems programmers to use large-computer-based implementation language processors and debugging systems as accepted tools of the trade. Already a compiler for the BLISS language, an implementation language developed at Carnegie-Mellon originally for the DEC system-10, has been produced to generate code for the PDP-11 on the DEC system-10.

THE MINI AS A BASE FOR DEDICATED SYSTEMS

Over the past five years (which represents virtually the entire history of the minicomputer in terms of number installed), there has been a noticeable shift in the hardware/software tradeoffs in pricing and using minicomputers in dedicated system applications. Several people have long maintained that the price of the software for a dedicated system should, as a rule of thumb, equal the price of the hardware. Although apparently true four years ago, this cannot possibly be true today. Hardware costs of minicomputer mainframes and memories have been decreasing exponentially at an apparent rate of about 25 percent a year, while software costs have been slowly rising, at the rate of inflation. Thus, if the hardware/software cost ratio was around 1/1 four years ago, it is more like 1/2.5 today, and the gap is steadily widening.

All of the extensibility promised by recent hardware developments, including the modular construction of the new machines and the increasing use of ROM's, should have an accelerating affect on the applications of minis to dedicated systems, steadily reducing the cost of the hardware required for a given task.

We see implementation languages beginning to relieve the trend toward higher software costs and hope they will continue to do so in the future. But we see problems in the high cost of training existing programmers to use implementation languages, the lack of acceptance by lower-level programmers, and general questions arising as to whether such languages and compilers really appropriately solve more difficult problems (such as those that are space and/or time critical).

We predict that in the next few years the current problems regarding implementation languages will be vigorously attacked on several fronts, and we feel reasonably certain that they will be increasingly used in dedicated systems, particularly those which are neither space nor time critical. For critical systems, we feel that some time will be required before compilers for implementation languages, and indeed the languages themselves, are improved and tested to the point that they can be of real value.

Several people, including one of the authors, have used interpretive languages to simplify the construction of dedicated systems. In this technique, an existing interpreter, for a language such as ESI[14] or FOCAL,[15] is modified so as to minimize the core requirements for the interpreter by removing unwanted features and adding additional ones. The resulting extended machine is thus an ESI machine, or a FOCAL machine; the user program in the higher level language is stored in memory and executed completely interpretively. John C. Alderman, in a paper presently in manuscript form, has referred to such languages as "plastic languages", in the sense that the systems programmer, in designing the dedicated system, can modify the interpreter for the language, and thereby change the virtual machine, in much the same way as writable ROM's can be used. Indeed, the two approaches can easily be combined; one could wire the interpreter in an ROM and plug it into the computer, thus creating an unalterable virtual machine for FOCAL or ESI.

It should be noted that the use of the "plastic language" technique is limited to those systems which are not time critical, since the interpretive nature of program execution makes the resulting virtual machine relatively slow. One of the authors has been quite successful in applying this technique to a proprietary dedicated system, where its advantages are quite significant particularly regarding ease of modification of the higher-level application code.

SUMMARY AND SOME SPECULATIONS

In closing, we would like to summarize a few points made earlier, and to exercise our license to speculate a little on the near future of minicomputers.

First, the user of minicomputers for the solution of general-purpose applications in data processing, scientific programming, or engineering, will find minicomputers increasingly indistinguishable from larger computers. With luck, he will not find that interactive control, which now distinguishes most minis from larger systems, has been thrown out in the wash.

Second, the cost/performance ratio of minicomputer hardware will continue to improve at the same rate as it has over the last five years.

Third, the minicomputer user will continue to receive the benefits of cost/performance ratios through decreases in cost for the same, or slightly improved, performance while the large computer user will generally continue to receive more performance at the same cost.

Fourth, and as a consequence of the above, all applications of minicomputers will become increasingly more economical. Thus, many applications which are not performed on minis today, or many which are not done on computers at all, will utilize minis in the near future as prices continue to drop.

It might be argued that time sharing could just as easily be used in the future to solve problems which do not use computers at all now. It is undoubtedly the case that time sharing will continue to be used for those problems which require its unique assets, i.e., essentially as much processor time, as much I/O, as much core and disk space as required, when you need it, purchased on an "as you use it" basis. But even at the most favorable rates available today, time sharing is *much* more expensive than using a mini. The lowest price we know of for time sharing is about $8 an hour for as much time as you can use, though this is on a relatively small system with quite heavy loading and a lot of contention.

On the other hand, even at today's prices, a mini can be bought for $5000. If this is written off over three years, and used 40 hours a week, an effective price of only about one dollar an hour can be approximated. A few years from now, this should drop to around 25 cents an hour.

We see the possibility of people providing a variety of "plug-in packages" for popular minis, quite likely software provided in the disguise of ROM hardware to provide the supplier with proprietary protection, product standardization, and integrity against unauthorized user changes. Some of these standard plug-in packages might be:

(a) The COGO virtual machine for the civil engineer.
(b) The JOSS® virtual machine for the engineer and statistician, replacing today's electronic desk calculators.
(c) The small business virtual machine, providing the retailer with a small machine capable of performing routine bookkeeping.
(d) The homemaker virtual machine, providing the busy housewife with a menu planner, household controller, alarm system, and checkbook balancer.

In conclusion, if the designers and product planners of minis think more clearly on what minis can do in both program creation and program execution, we may see an end to the minicomputer syndrome.

> *"He nodded and went out . . . and in a moment I heard Winnie-the-Pooh—bump, bump, bump—going up the stairs behind him."*
>
> Ending of *Winnie-the-Pooh,* A. A. Milne

REFERENCES

1 A W HOLT W J TURANSKI
Extended machine—Another approach to interpretive storage allocation
Under Army contract DA-36-039-sc-75047 Moore School of EE U of Pa 1960

2 C J SIPPLE
Computer handbook and dictionary
Bobbs-Merrill Indianapolis 1966

3 R W WATSON
Timesharing system design concepts
McGraw-Hill New York 1970

4 *PDP-7 CORAL manual*
Applied Data Research Inc Princeton N J 1967

5 *Broadcast programmer user's manual*
Applied Data Research Inc Princeton N J 1968

6 *Interdata microprogramming manual*
Interdata Ocean Park N J

7 A KOTOK
DEC debugging tape
Memo MIT-1 rev MIT December 1961

8 D J WAKS
Interactive computer languages
Instruments and Control Systems November 1970

9 D J WAKS
The MIMIC programming automation system
Applied Data Research Inc Internal memorandum July 1971

10 *MIMIC user's guide*
Applied Data Research Inc Ref #01-70411M May 1971

11 *AED manual*
Softech Cambridge Massachusetts

12 *BLISS-10 manual*
Carnegie-Mellon Univ Pittsburgh Pa

13 *ESPOL manual*
Burroughs Corp Detroit Mich

14 D J WAKS
Conversational computing on a small machine
Datamation April 1967

15 *FOCAL user's manual for PDP-8*
Digital Equipment Corp

The current state of minicomputer software

by JOSEPH F. OSSANNA

Bell Telephone Laboratories, Inc.
Murray Hill, New Jersey

INTRODUCTION

No one who has watched the spectacular growth of the minicomputer industry needs to be told that people are finding new small computer applications at an astonishing rate. Undoubtedly there are many reasons for this striking increase in minicomputer utilization. Certainly one reason is the increasing sophistication of minicomputer software. This paper will summarize the development of this sophistication, and will try to characterize its present state. Minicomputer hardware evolution will be discussed because of its effect on minicomputer software evolution. The range of minicomputer uses and users will be characterized along with their varying software organizations. Commentary on techniques of minicomputer software development is included. A number of examples of both past and present minicomputer software systems are presented.

MINICOMPUTER HARDWARE

A minicomputer is a "minicomputer" by nature of its hardware which is relatively simple, physically small, and comparatively inexpensive. This is accounted for by the following general characteristics compared to the "ordinary" large computer.

- Shorter wordlength
- Smaller memories
- Fewer and simpler instructions
- Simpler input/output capabilities
- Minimal features
- Fewer peripherals

In addition to the above, limited software support by the manufacturer and lower marketing costs have contributed to the lower price of the minicomputer. Wordlengths range from 8 to 24 bits with 16 bits being the most common in currently available machines. The range of memory sizes available in a typical current machine is from 4K words (K=1024) to 32K words. Basic instruction sets typically lack multiply and divide, decimal and floating-point arithmetic, arbitrary shifting, and fancy character string manipulation. Although minicomputer memories are smaller than those of large machines, the memory cycle times frequently are similar, with the result that computing speeds are similar in applications where a smaller word size and the absence of some types of arithmetic are not a factor. Basic input/output structures usually employ a single time-shared bus with limited device addressing and with limited parallelism during data transfer. Features such as memory protection, relocation, and segmentation are usually nonstandard or unavailable. Often few fast registers are included. The selection of peripherals standardly interfaced to a given machine typically has been limited. In the past, the only reliable peripherals available have been expensive ones originally designed for the larger computer systems.

To some extent, what has just been said above is becoming less true with time. The spectacular advances in integrated circuit technology in recent years have resulted in dramatic price declines. Core memory and peripherals have also become less expensive. The price of a basic system consisting of the central processor (cpu), 4K×16 bits of core memory, and a Model 33ASR teletypewriter has dropped from about $25,000 in 1966 to about $8,000 in 1971.[1] The circuit technology advances have also made increased system capability economically attractive. The following amendments to the list of minicomputer characteristics are presently apropos.

- Instruction sets improving; fast multiply, divide, and floating-point optionally available. More and better addressing modes being designed.
- Memory protection, relocation, and segmentation becoming optionally available.

- Larger sets of fast general registers becoming common.
- New bus concepts are providing better input/output structures.
- Priority interrupt structures and interrupt vectors are becoming standard.
- Smaller and well designed peripherals are available at reasonable cost.
- Less expensive disk storage is becoming available in sizes commensurate with small machines.

Manufacturers have achieved this and are providing increased software support without noticeably stemming the price decline.

These recent improvements in small machine architecture have attracted more sophisticated applications and more sophisticated systems designers. The advent of inexpensive disk storage is probably the most important occurrence influencing minicomputer software evolution.

A tabular summary of currently available minicomputers and their characteristics can be found in Reference 2. Similar information along with projections of the future of the minicomputer industry are given in Reference 1.

MINICOMPUTER USES AND USERS

Most minicomputer systems consist of a general-purpose small machine configured in a way suitable for some particular application. Tailoring the size and complexity of a system to the requirements of a particular job can result in an economy of specialization that outweighs the economy of scale of using a part of a larger machine.[3,4] Accordingly, one would expect the range of applications to break down in a way that reflects this economic bias. Auerbach[1] reports that in 1970 minicomputer dollar sales were divided as follows.

45%	Industrial and process control.
3%	Peripheral control (COM, OCR, line printers, etc.).
20%	Communications control (concentrators, switching).
10%	Computations.
22%	Data acquisition and reduction.

Process control and data acquisition often involve intimate connection to distributed sensors or laboratory instruments. This is often infeasible for centrally located large machines, especially when coupled with the necessity for rapid control by the user and his program.

In a survey conducted within Bell Laboratories in the Spring of 1971 of approximately 200 minicomputer systems the following usage breakdown was reported.

31%	Computerized testing of equipment and devices.
19%	Data acquisition and reduction.
18%	General-purpose computing, some with data acquisition.
15%	Peripheral control (plotters, cameras, displays, unit-record equipment).
9%	Intelligent graphic terminals and interactive graphic systems; general-purpose graphic-oriented computing.
7%	Process control.

It is possible to observe different software organizations across these various types of users. Some of the differences are clearly due to the nature of the applications. Many are due to the nature of available software support, to the software experience of the programmers involved, and to hardware limitations. In the next section these differences are characterized.

SOFTWARE ASPECTS

The choice of a software organization for a minicomputer system can profoundly affect hardware requirements, general efficiency, the range of applications, extensibility, and the ease of software maintenance and modification. The best organization in a particular case depends on which considerations are paramount. Other considerations include the following.

- Are the application programs to be run under an operating system?
- Is there to be on-line secondary storage?
- Are there crucial real-time response requirements?
- Are there to be multiple simultaneous users?
- Is the system to be self-supporting—i.e., can the application programs be developed and maintained on the system?
- Is there a justification or need for core protection, relocation, or some sort of virtual memory?

An "operating system" is a collection of basic and frequently used utility programs essential for convenient and efficient use of a machine. For many people this loose definition means the sum of all the programs they use that were written by someone else and consequently not easily modified. Most minicomputer operating systems are simple and basic compared to those used on large machines. The most basic form of operating system consists of programs to perform physical input/output and handle code and/or format conversion of the transmitted data, and a program to bring the user's program into core and give control to it. If the minimization of core usage is paramount, the generality of even a basic operating system may need to be foregone and only the absolutely necessary input/output programming be included directly in the user's program. Such inclusion of operating system functions in application programs is not uncommon on small machines.

If the same machine is to be used to develop the user's programs, other support programs are needed. An editor is needed to create and modify the user's source text (unless he is doing it manually, for example using cards). An assembler or compiler is needed to translate this source text into executable machine code. Optionally a debugger is useful for analyzing program troubles. Operationally, the first two are run like user programs, and the latter is usually combined with the user program if needed.

If there is justification for on-line random-access secondary storage such as a disk or drum, a different perspective is possible. The less frequently used portions of any operating system and of the user's program need not be permanently core resident, and core requirements can be minimized without foregoing the convenience and generality of an operating system. Further, support programs like those mentioned above can be stored on disk for handy use (by disk will usually be meant any such secondary device). Additional support in the form of libraries of user and utility programs can be easily kept along with a binder program that can search libraries and combine collections of program modules into executable units. Greater convenience is obtained when the operating system provides organized access to the disk by means of some sort of a file system. A well designed file system is usually the foundation of the better general-purpose operating systems.

Real-time response requirements can have strong implications for software system design. By real-time we do not here mean the requirements of controlling conventional peripherals such as disks, tapes, unit-record equipment, or most communication lines. In these cases a too-slow response usually causes retries and degrades efficiency but is not ordinarily fatal. What is meant are the response requirements of controlling such things as a steel rolling mill, a linear accelerator, or an oil refinery. Industrial process control systems typically receive a variety of external stimuli requiring a wide range of response times. This necessitates a hierarchical priority structure for the interruption and execution of programs. When there is more than a few such programs the scheduling can become quite complex. Operating systems for such situations are usually different than general-purpose ones; process control systems are discussed later. Data acquisition systems and communications control systems can have real-time response problems, if peaks can occur that would temporarily tax their processing capacity and it is important that no data be lost.

Unlike a process control system which supervises known external processes with generally known chari acteristics, a multi-user system such as a general-purpose time-sharing system faces user submitted tasks of more-or-less unknown characteristics. For example, resources such as processor time, core space, and file system space demanded by the user processes vary dynamically, and the system generally must schedule adaptively and enforce various limits on resource utilization. Even if a minicomputer system is economically justified when wholly committed to some application, a multi-user capability can permit software development to take place while the main application is running. An example of a multi-user system is given later.

Core protection is a hardware feature that enables the operating system to make sensitive regions of core storage inaccessible, thereby protecting those regions from damage by being written or preventing information from being read. This feature is uncommon in minicomputer systems that operate with debugged programs, but is usually mandatory in multi-user systems that permit users to write and execute arbitrary programs. Relocation is a hardware feature that permits a program to be loaded at any convenient place in core memory, and to be moved (while interrupted) to another place. This feature can be a great asset to, for example, a process control system trying to juggle a myriad of different sized programs between disk and core. Virtual memory[5] is a concept where a program can think it has more core than it really has or even than there is.

RETROSPECTION

When one looks at the minicomputer systems being used in the early 1960s one sees minimal hardware with typically no disk storage and minimal software support in the form of operating systems, compilers, editors, etc. The temptation was therefore great to make use of the nearest computer center with its big machine, big disk, and varied software support.

One of the first minicomputer applications that the author came in contact with at Bell Laboratories circa 1964 was an experimental remote graphical display system called GRAPHIC-1.[6] The hardware consisted of a Digital Equipment Corporation (DEC) PDP5 computer with 4K×12 bits of core, a modified DEC Type 340 display scope, a 4K×36 bit display memory, a light pen, Model 33ASR teletypewriter, and a card reader. In addition there was a high-speed connection to the direct data channel of a nearby IBM 7094. Support software consisted of an assembler and postprocessor to assemble, merge, and link relocatable programs to form executable units for the PDP5, a display material assembler, and a library of input/output and inter-computer communication programs. These support programs ran on the 7094 and produced cards that were read into the PDP5. End users of the system could write in a higher-level graphical language called GRIN[6] which also ran on the 7094, and could use the system to submit a regular batch job to the 7094. This dependence on the local big machine was a natural yielding to the temptation to do software development in a convenient environment.

GRAPHIC-2[7], a successor system to the one above, emerged in 1967 and is notable for its relatively sophisticated software. It is implemented on a DEC PDP9 computer with 8K×18 bits of core, a fast fixed-head disk, display, light pen, fast paper tape reader/punch, and a dial-up 2000 bit/sec link to the local big machine. As in the preceding case all the software for the PDP9 was written using support programs on the local computing center large machine, a Honeywell Information Systems 635. The end user is provided with a graphical programming language called GRIN-2[7] and an assembler, both of which run on the 635. In the next version of the system these will run on the PDP9 itself. GRIN-2 provides the ability to control the man-machine interaction, to create and edit graphical data structures, and to control the display. The system may be used in a stand-alone mode or it may be connected to the 635 and used in a mode where programs executing in the 635 send over programs or data on demand to the PDP9. One of the more interesting aspects of the software in this system is that it provides a form of virtual memory to the user program. To do this, each user's program and data is broken up into blocks which are assigned unique identification numbers (IDs). All references between blocks are in terms of this ID and an offset within the block. When a block is referenced it is retrieved from disk or possibly from the big machine. Only the currently needed blocks are actually in core at a given time. The user is relatively free from worrying about the size of core; one user application program consists of nearly 60K PDP9 words.

Another example is a small laboratory computer system used for the acquisition and reduction of data from a spectrometer. It originally consisted of a DEC PDP8 with 4K×12 bits of core, a Model 33ASR teletypewriter, a DEC Model 34D display, a home-made analog-to-digital converter, an extended arithmetic element, and interfaces to laboratory equipment. The user wrote his own application software in a few months using a DEC supplied editor and assembler on the PDP8 itself. Both user and support programs were stored on paper tape and had be be loaded using the reader on the 33ASR. How long does it take to read 4K×12 bits at the rate of 10 8-bit characters per second? This user eventually got 128K×12 bit drum primarily to facilitate the acquisition of more data than would fit into core. He now uses a DEC supplied disk-based operating system for program development and to run his application. This system is typical of quite a few minicomputer systems within Bell Laboratories.

A more recent (1968) example of a laboratory computer system designed for self-service by a variety of users at Bell Laboratories is described in Reference 8. The hardware consists of a Honeywell DDP224 with floating-point instructions, 32K×24 bits of core, two disk drives for 2.4 million word removable disk packs, magnetic tape, a card reader, analog-to-digital and digital-to-analog converters primarily for speech and music, a console typewriter, various graphical input and output devices, and miscellaneous devices such as a piano keyboard. Extensive software support is provided for the user in the form of a simple disk operating system with utilities for all the various devices, an assembler, a Fortran compiler, an editor, a loader, and a large library of useful programs. A simple file system is provided along with disk utilities for physical disk accessing. Disk conflicts are avoided by informal agreements and by many users having their own disk packs. The operating system will bring whatever program the user wants into core from disk but the user must manually transfer control to its starting location. This machine has been very successfully used by more than a dozen users for various research projects. A main drawback is the necessity for users to sign up for time on the machine, and a limited multi-user capability would undoubtedly be welcome. The system is much like a user-operated batch processing system. Even though some of the applications involve real-time response and are demanding of the processor, such tasks as file editing and compiling could go on at a lower priority or be done concurrently with less demanding tasks. Such an ability would require a better file system for user isolation and some scheme for rotating user programs through or sharing core. In this system, such additions would be tantamount to a complete redesign.

INTROSPECTION

Just as in the case of software design evolution on big machines, the evolution of software on small machines has not been homogeneous throughout the field. The inertia inherent in the use of successful software can last for times comparable to the current age of the minicomputer era. Most of the systems described earlier are still working. There probably isn't any early software design or development techniques that isn't still being practiced today. The major impetus for software design improvement comes from the introduction of more advanced machines and from the infusion of fresh software techniciars. Attracted by the newer small machines and often experienced on big ones, these fresh workers are beginning to achieve on the minicomputer a decree of software design sophistication previously seen only on larger machines.

In the next section an attempt will be made to characterize minicomputer software currently in common use and to describe some forefront efforts.

CURRENT EXAMPLES

Industrial process control

All indications are that this field will be one of the largest users of minicomputers.[1] Work in this field, not all involving minicomputers, is relatively extensively documented.[9,10] As indicated previously, process control applications often involve a wide range of real-time response requirements. Control algorithms involve the operation of complex multi-loop negative feedback systems. Most systems involve two forms of control: direct digital control in which the computer directly controls process variables; and supervisory control in which the computer sets the operating points of otherwise independent analog controllers. The response required in direct digital control ranges from milliseconds to seconds, while that required for supervisory control ranges from seconds to minutes. In addition, there is usually an overall optimizing and adaptive control function that slowly modifies the control algorithms. On top of this response time hierarchy can be management information and long-term scheduling functions involving delays measured in days or weeks. Direct digital control tasks are usually many in number but relatively simple while supervisory control tasks are fewer in number but relatively complex.

The number of task-oriented programs is usually sufficiently large that economy dictates that they not all be core resident; thus programs need to be brought into core when needed. The existence of the shorter real-time requirements as well as the desire for reasonable efficiency dictates that several programs reside simultaneously in core. Thus a process control system takes on the appearance of a multi-programmed time-sharing system. In contrast with time-sharing systems, the scheduling of the execution and interruption of programs is more complex in process control systems.[10] The relative priority of programs is typically dependent on process conditions, operator intervention, and even such things as the time of day. While the general technique is to respond to hardware interrupts and place programs into execution priority queues, provision must usually be made for certain high priority programs to purge certain other low priority programs from queues and for low priority programs to inhibit their interruption at certain critical times. Some sort of organized provision is required for the interlocking of shared data bases. Although most of the programming is done in assembly language and in Fortran, a number of process control oriented languages have been developed and used.[11] The process control field has in some ways reached a greater level of sophistication than other fields of application.

Communications control

This field includes communication line concentrators and controllers used as communication front-ends for large computer systems, as nodes in communication networks, and as message switching centers. The latter application requires secondary storage while the others do not. Data line concentration and control can usually be done less expensively and more flexibly with a minicomputer than with wired hardware controllers.

An example of a minicomputer front-end is the line concentrator used to interface varied terminal equipment to the Michigan Time-Sharing System.[12] This system was experimental and its hardware complement is not necessarily optimized. It consists of a DEC PDP8 with 12K×12 bit of core memory, a Model 33ASR teletypewriter, a high-speed paper tape reader/punch (to facilitate software development), an extended aiithmetic element, an interface to the local IBM 360/67's multiplexor channel, and interfaces to standard telephone data sets operating at all the standard speeds. The lower speed interfaces are bit buffered and the faster ones are character buffered. The system performs a limited form of terminal type identification of low speed asynchronous typewriter terminals and handles a variety of IBM and teletypewriter terminals. Because the speed of operation of each line is program controlled, statistical sharing of the access lines reduces the required number of telephone ports. The system-terminal interface is better human engineered than that obtained using standard IBM hardware line controllers. Full duplex operation of teletypewriters is standard and read-ahead of the terminal keyboard is provided.[13]

While such a system is less complex than many process control systems, the bit and/or character completion interrupt rate is high and the responding programs can't dawdle. The low number of distinct tasks combined with the high task rate dictates that the software all be core resident. The system described above was equipped to serve 16 low speed asynchronous lines and 4 voice grade synchronous lines; however, a system of this kind more typically can serve between 32 and 64 low speed lines along with several voice grade lines.

Paper tape operating systems

The above heading may overdignify the collection of software that it refers to, but this kind of software support is one the oldest and is still common. If one has a typical basic minicomputer configuration consisting of a processor, 4K words of core, and a teletypewriter with paper tape handling, one either does his software development using support programs stored on paper tape or uses some other machine for development and reserves the basic machine to run the application. The main drawback with paper tape based support is the slow rate of reading and punching paper tape. This drawback can be largely mitigated by adding a high speed reader/punch. Surprisingly enough, the convenience of working on your own machine and the fact that a cycle of editing, assembling, and reloading may be shorter than the time required to travel to and wait on the local computer center, may make the use of a paper tape operating system feasible. Of course, access to a good time-sharing system with the right kind of program support or to another machine like yours but bettei equipped is probably preferable.

An example of a current paper tape system in DEC's paper tape software for the PDP11.[14] It provides an absolute loader for reading in all programs, a reasonable text editor, an assembler, floating-point simulation, a math package, a collection of input/output utilities, and an octal debugger. The editor is used to create and edit programs at the teletypewriter; during editing, the old copy of the source is read off paper tape a block at a time, edited, and then punched. Assembling a program consists of reading in the assembler, feeding the source program to the assembler twice, and receiving an executable object copy of your program from the punch. Your program can then be tried by loading it; after loading the loader will transfer control to it automatically. If it works, fine; if not the octal debugger can be loaded into vacant core (if any) and used to poke around in the remnants. All of this is or is not as awkward as it seems depending on the alternatives open to you. With enough core a simple core resident operating system containing all the support programs with some core left for the user would be even more convenient; the additional cost is comparable to the cost of the smaller disks.

Disk operating systems

It has been observed several times previously that the addition of disk storage to a minicomputer system makes a world of difference. Even the simplest disk-based operating system greatly magnifies the convenience of working on your own machine, which becomes like using a one-user time-sharing system. Multi-user systems are possible and desirable and are discussed later. Much of what is possible with disk is possible in a slower more restricted way using randomly accessible magnetic tapes such as DEC's DECtape,[15] or even small tape cartridges.

An example of a recently available (circa 1968) disk operating system for a machine that has been around in some form for awhile is the PS/8 operating system for the PDP8 family of machines.[16] It requires 8K words of core and a disk *or* DECtape.

Another example is DEC's disk operating system for the PDP11.[17] It requires at least 8K words of core, a small disk augmented by DECtape, or a larger disk, a high speed paper tape reader/punch, and a console typewriter. The support software includes a command interpreter, a reasonable text editor, an assembler, a Fortran compiler, an octal debugger, a file utility package, floating-point simulation, a math package, and a linker to combine programs. The operating system provides a reasonable file system and does input/output for the user. Core is divided into five regions: a small region at the bottom containing the interrupt vectors which indicate the location of interrupt handling programs; next a small area for the resident monitor; an area for the user program at the top of core; below that the stack (where parameters are stored temporarily during the transfer of control between programs); and a free area divided into blocks and used for buffers, temporary tables, and device drivers brought into core.

To invoke a system support program such as the editor or a user program written to run under the system, the user types a request on the console keyboard which can include information about which input and output devices are to be used during the run. User programs can be used interactively and can be interrupted for debugging.

Multi-user time-sharing systems

It is an accepted fact nowadays that the time-shared use of a computer is one of the more effective ways of increasing the productivity of a group of users in need of computing service. The advantages of "working on your own machine" are effectively extended to the whole group at once. A multi-user system is inherently more complex than a single user one; it is necessary to multiprogram and/or swap user programs between disk and core and to keep track of everyone's input/output and machine conditions. It is helpful too, if users can be prevented from injuring each other and the system.

We shall present here, as an example of a multiuser system, a system recently developed (primarily by K. L. Thompson) at Bell Laboratories which is felt to be at the forefront of minicomputer software design. This system utilizes a DEC PDP11/20 computer with 12K×16 bits of core storage, a 60 cycle clock, a 256K work fixed-head disk, a disk drive for 1.2 million word disk cartridges, a dual DECtape drive, 8 low speed asynchronous line interfaces for remote typewriters, and a little-used paper tape reader/punch.

Core is divided into an 8K word/region containing the totally core resident operating system and a 4K word region for the user program and his stack. The system space includes about 2K used for a pool of dynamically allocated buffers. The operating system maintains an excellent file system, creates and manages user processes executing core images), does input/output for the user, and performs all scheduling functions. Scheduling is by placing ready-to-run users on one of three queues, depending on whether the user has interacted (has just typed an action character on his console), some pending input/output has completed, or the user has run out of time allotted for execution. All users are swapped between the fixed-head disk and the same region of core.

The main feature of this system is a versatile, convenient file system with complete integration between disk files and all input/output devices. There are three kinds of files; ordinary disk files, directories, and special files representing physical devices. A directory is like an ordinary file except that only the system can modify it. The file system is a tree structured hierarchy originating at a root directory. Any type of file can occur at any level. Files have names of eight of fewer characters. At any given time a user process is associated with a particular current directory. When the user program wishes to open or create a file, it gives the system the file name of a file in the current directory or it gives a path name which either specifies the absolute location in the tree or the location relative to the current directory. If the reference is successful, the system returns a file identification number to be used for future references. File system protection consists of associating with the file at time of creation the name of the creator and permitting him to specify whether he and others can read and write the file. Files are formatless and are regarded as strings of 8-bit bytes.

Another feature of the system is the ability to initiate asynchronously executing processes from within a program or from command level. Commands are normally terminated with semicolons or new-lines; the command interpreter creates a process to execute the command and waits for its completion. If the command is terminated instead by an ampersand, the same thing occurs except that the command interpreter doesn't wait. Argument strings typed with a command are made available to the program by placing them on the stack.

The name of any executable program can be used as a command. The name is first searched for in the current directory and if that fails it is then searched for in a system library. A myriad of commands are available including those for listing directories, moving, copying, and deleting files, changing the owner and mode of a file, and changing the current directory. Other programs available include a context text editor, an assembler, a symbolic debugger, a linking loader, a dialect of Basic, Fortran, a text formatter, games such as chess, and various experimental languages.

The system operates 24 hours a day, seven days a week. A user community of about a dozen people access the machine regularly although it is not a service offering. The group that owns the machine uses it heavily for preparing documentation using the editor and text formatter. The maximum of eight simultaneous users (limited by the number of lines) does not tax the machine's capacity. Because of the lack of core protection, any user can write, assemble, and run a program that could damage the system and cause a system crash. However, since most users use debugged programs like the editor or text formatter, and because those few who develop new programs are careful, surprisingly few crashes occur. A future version of this operating system will be developed for a version of the PDP11 having core protection and relocation.

Virtual memory systems

One of the more interesting areas in minicomputer software is the use of virtual memory to manage the core and secondary storage of a small machine. A system in an earlier example, GRAPHIC-2, supported a form of virtual memory without the use of special hardware. Another similar example is a small time-sharing system on a Honeywell DDP516[18]. A different example involves the use of hardware assisted segmentation and paging on a specially modified Interdata Model 3[19].

CONCLUSIONS

There has been a decided evolution toward greater sophistication in the design and use of minicomputer software during the relatively short years of the minicomputer era, although many of the earlier methods are still in use. A major force toward software improvement has been the introduction of new and better machines and the advent to relatively inexpensive disk storage. The expanding minicomputer field has benefited from the attraction of fresh software design talent often with large machine experience. Some recent forefront efforts are yielding minicomputer operating systems with features previously found only on large machine systems.

REFERENCES

1 *Minicomputers*
Auerbach Technology Evaluation Series EDP Series No 4
Auerbach Info Inc Philadelphia Pa 1971

2 D J THEIS L C HOBBS
The minicomputers revisited
Datamation Vol 17 No 10 pp 25-34 May 15 1971

3 M V MATHEWS
Choosing a scientific computer for service
Science Vol 161 p 2368 July 5 1968

4 J R COX JR
Economies of scale and specialization in large computing systems
Computer Design November 1968

5 P J DENNING
Virtual memory
Computing Surveys Vol 2 No 3 pp 153-189 September 1970

6 W H NINKE
Graphic-1—A remote graphical display system
AFIPS Conference Proceedings Vol 27 Part 1 1965 FJCC
Spartan Books Washington D C pp 839-846

7 C CHRISTENSEN E N PINSON
Multi-function graphics for a large computer system
AFIPS Conference Proceedings Vol 31 1967 FJCC
Thompson Books Washington D C pp 697-711

8 P B DENES M V MATHEWS
Laboratory computers: Their capabilities and how to make them work for you
Proceedings of the IEEE Vol 58 No 4 pp 520-530 April 1970

9 *Special issue on computers in industrial control*
Proceedings of the IEEE Vol 58 No 1 January 1970

10 C L SMITH
Digital control of industrial processes
Computing Surveys Vol 2 No 3 pp 211-241 September 1970

11 H E PIKE JR
Process control software
Proceedings of the IEEE Vol 58 No 1 pp 87-97 January 1970

12 D L MILLS
Preprocessors in a data communication environment
Proceeding of ACM Conference on Problems in the Optimization of Data Communications Systems
Pine Mountain Ga pp 286-310 October 1969

13 J F OSSANNA J H SALTZER
Technical and human engineering problems in connecting terminals to a time-sharing system
AFIPS Conference Proceedings Vol 37 1970 FJCC
p 355-362

14 *PDP11 paper tape software programming handbook*
DEC-11-GGPA-D Digital Equipment Corporation
Maynard Massachusetts 1970

15 *PDP11 peripherals and interfacing handbook*
Digital Equipment Corporation Maynard Massachusetts 1971

16 *PS/8 system user's guide*
DEC-P8-MEFA-D Digital Equipment Corporation
Maynard Massachusetts 1970

17 *PDP11 disk operating system*
DEC-11-SERA-D Digital Equipment Corporation
Maynard Massachusetts 1971

18 C CHRISTENSEN A D HAUSE
A multiprogramming, virtual memory system for a small computer
AFIPS Conference Proceedings Vol 36 1970 SJCC AFIPS Press Montvale New Jersey pp 683-690

19 B H LISKOV
The design of the venus operating system
Proceedings of the Third ACM Symposium on Operating Systems Principles Stanford University pp 11-16 October 1971

Real-time fault detection for small computers*

by J. R. ALLEN

Bell Telephone Laboratories
Naperville, Illinois

and

S. S. YAU**

Northwestern University
Evanston, Illinois

INTRODUCTION

Advancing technology and declining costs have led to a sharp increase in the number and variety of small computers in use. Because small computers are readily suited for many real-time applications, a great deal of work has been directed toward simplifying the interface between the computer and its peripherals. Hardware interrupting capability and a specially designed I/O bus are required for peripheral device interfacing in a real-time environment and such things as direct memory access, data channels, and multilevel hardware and software interrupt capability are common. These machines tend to be parallel, synchronous computers with a relatively simple architecture.

In a real-time environment, fault detection can be of major importance. Much of the past work has been directed toward the design of additional hardware, internal to the computer, which allows critical feedback loops to be controlled and often inserts special registers for the maintenance task.[1-4] These techniques require that the maintenance circuitry be designed concurrently with the computer itself and have access to components internal to the computer. Many problems can arise from attempting to modify an existing computer. For example, critical timing and gate fan-out can be disturbed, and most warranties become void if unauthorized modifications are made to a computer.

Other techniques cannot be used for real-time fault detection because they require manual intervention, excessive storage, noncontiguous memory locations or excessive execution time.[5-7] Much of the previous work attempts to locate faults as well as detect them. While fault location is desirable, it is more expensive and requires much more time and storage than fault detection. Many applications do not require fault location. It may be more feasible to either interrupt the task or perform the task manually during the diagnosis and repair interval; the most important thing is to recognize that the computer may be performing the task incorrectly.

An earlier technique attempted to simulate each instruction using different instructions and comparing the simulation result with the actual result.[7] This technique requires that the computer be somewhat operational and that it be capable of comparing the two results. A means is needed for determining that the test routine is executed periodically.

In this paper, we propose a real-time fault detection scheme for small computers which is effective for faults in the central processor unit (CPU), core memory and I/O bus. It requires an external monitor which is simple, inexpensive, and interfaces to the computer's I/O bus just as any other peripheral device. The monitor periodically triggers a program interrupt, causing the computer to execute a predefine test routine. During the course of the routine's execution, several key bit configurations are transmitted to the monitor. If the computer fails to respond or if the bit configuration does not match the one that is expected, then the computer is assumed to be faulty and the device may either cause a power down or sound an alarm.

The proposed technique compares favorably to the previously referenced techniques in fault detection. Certain faults will not be detected, however, because

* Portions of this Work were supported by U. S. PHS Grant No. 5 P01 GM 15418-04.

** Depts. of Electrical Engineering and Computer Sciences, and Biomedical Engineering Center.

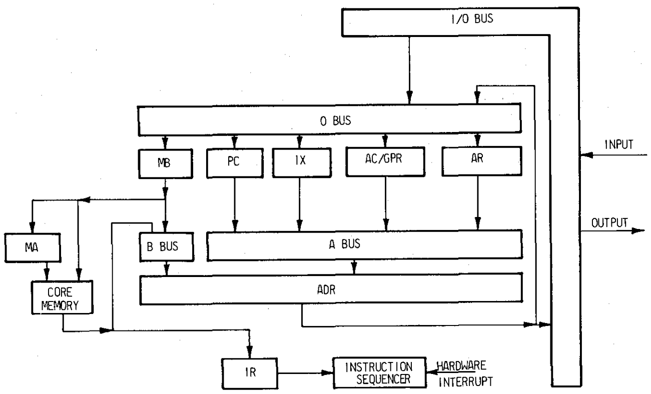


Figure 1—Typical computer architecture

internal modification to the computer is not allowed. In the past, real-time fault detection has carried a price tag comparable to the cost of the computer itself. A major advantage of this technique is that its low cost makes a great deal of fault detection possible in applications which would not previously have been cost effective. The fact that the technique assumes that no modification may be made to the computer means that many existing systems could make use of the method without extensive modifications.

COMPUTER ARCHITECTURE

The proposed fault detection method may be used with a variety of small computers, all of which are very similar in architecture. Included in this class of computers are such machines as Digital Equipment Corporation's PDP-8, PDP-9/15; Hewlett-Packard's HP-2100; Honeywell's DDP-516; Data General's Nova-1200 and Nova-800; Xerox Data System's Sigma-3; and Hitachi's HITAC-10. All of these are parallel, synchronous machines with hardware interrupt and I/O bus.

Figure 1 shows a block diagram of a typical small computer. The *program counter* (PC) contains the address of the next instruction to be executed. The *memory buffer register* (MB) serves as temporary storage for information being written into and read from memory. The address of the memory cell to be accessed is placed in the *memory address register* (MA). The decoded contents of the *instruction register* (IR) controls the instruction sequencer. Other registers commonly found are an *accumulator* (AC), *general purpose registers* (GPR), an *arithmetic register* (AR) and an *index register* (IX). Most small machines make use of some subset of these registers.

Most of the logic functions are performed in the general purpose logic unit (ADR). A pulse on the hardware interrupt lead will cause the instruction sequencer to save the appropriate registers and to begin executing a different program.

ADDITIONAL HARDWARE REQUIREMENTS

This section explains the function of each portion of the block diagram for the monitor hardware shown in Figure 2.

The DEVICE SELECT modules (DS1 and DS2) are basically an array of AND gates which allow the peripheral device control signals and data signals to pass only when the proper device select code is present on the device select bus. This allows several peripheral devices to share the same I/O bus without interfering with one another. Three device codes are required, XX, $\overline{XX}$, and YY. It is desirable to have one device code ($\overline{XX}$), which is the complement of the other to verify that each bit of the device select bus can be set to both logical states.

Device codes XX and YY cause the WIRED REGISTER to be gated onto the Input Bus. The WIRED REGISTER is constructed simply by attaching alternate bits to a voltage representing a logical one and the remaining bits to logical zero. When device code YY is selected, the "DEVICE = YY" lead is enabled causing the contents of the WIRED REGISTER to be complemented before being placed on the Input Bus. Reading from both device XX and device YY causes both a 1 and a 0 to be read from each bit position. Many computers determine the status of a peripheral device by executing an instruction which causes the next instruction to be skipped if a status FF is set. By executing this instruction with device code XX the BUSY/IDLE FF will appear to be busy; if device code YY is used, the "DEVICE = YY" lead causes the FF to appear to be idle. In this way the device status testing feature may be checked.

Device code $\overline{XX}$ is used when outputting bit configurations to the monitor for comparison to the expected value. The INTERRUPT TIMER is simply a monostable FF which, after an appropriate delay, will cause a program interrupt to be sent to the computer and at the same time sets a FF, which enables the RESPONSE TIMER. The testing frequency is set by adjusting the monostable's delay time.

The RESPONSE TIMER is used to determine that the computer is taking too long to respond and may be "lost." If the RESPONSE TIMER is not reset or disabled before it "times out," an OVERTIME signal

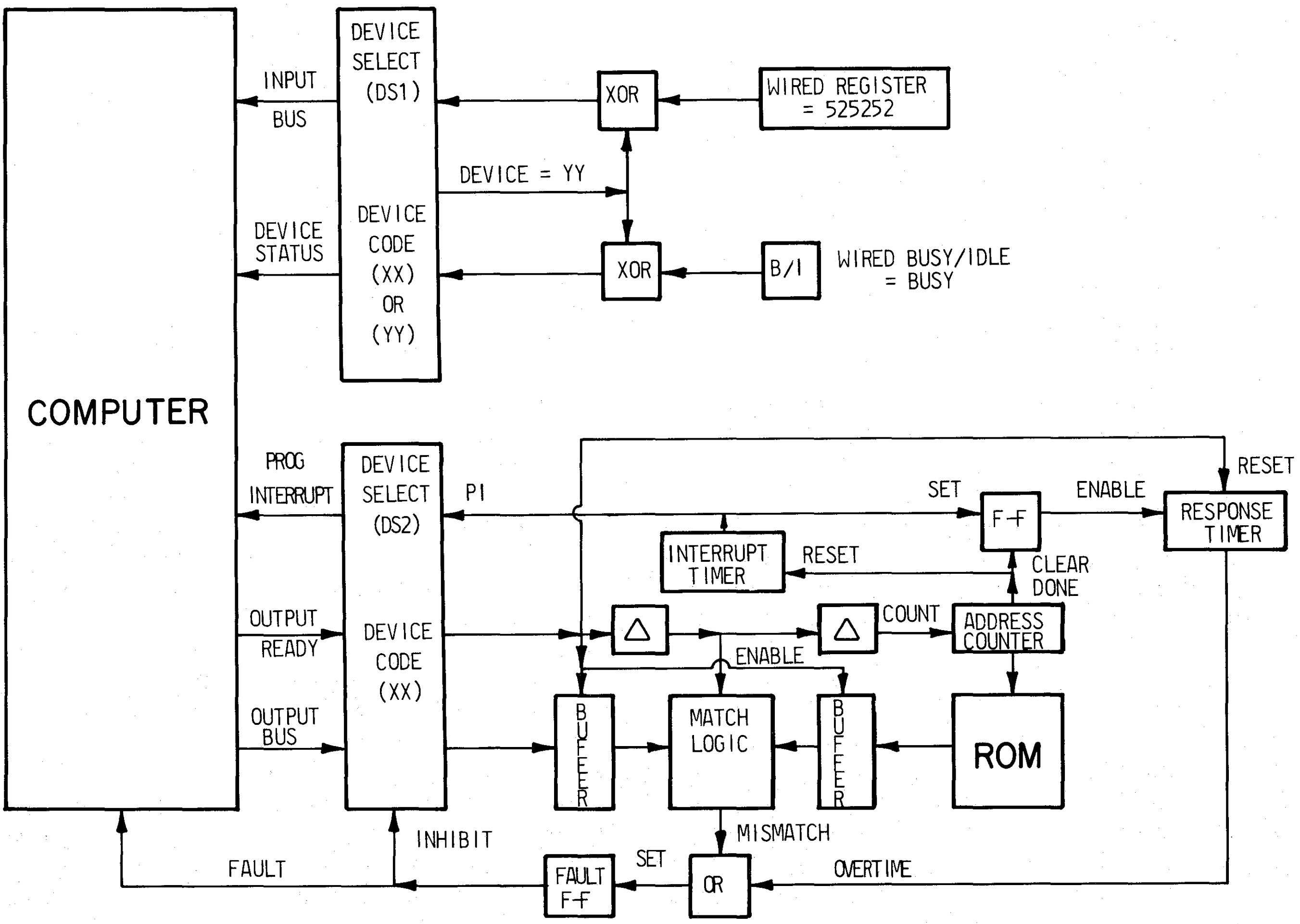


Figure 2—Block diagram for monitor hardware

is generated, indicating that a fault has been detected. A circuit to perform the RESPONSE TIMER function is shown in Figure 3.

The ADDRESS COUNTER is simply a cyclic counter designed to count modulo N, where N is the number of responses expected during a normal test. When the counter resets, a DONE signal is generated which disables the RESPONSE TIMER and resets the INTERRUPT TIMER. The ADDRESS COUNTER is incremented after each output from the computer to sequentially address the contents of the read-only memory (ROM). The ROM need not be large and may be built economically from a diode array.

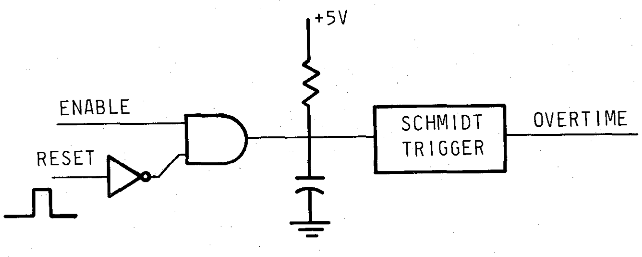


Figure 3—Response timer

The response from the computer and the expected response, read from the ROM, are both buffered and compared by the MATCH LOGIC. When enabled, the MATCH LOGIC basically OR's together each bit of the XOR of the two buffers to produce the MISMATCH signal. An OUTPUT READY signal from the computer is used to load the BUFFER, enable the MATCH LOGIC, reset the RESPONSE TIMER, and increment the ADDRESS COUNTER, all after appropriate delays.

If either a MISMATCH or an OVERTIME signal is produced, the FAULT FF is set. This inhibits any further output to the monitor, thus preserving the contents of the ADDRESS COUNTER and the two buffers as an aid for the diagnosis of the fault.

The FAULT signal may be used in whatever way seems appropriate; it may stop the computer, sound an alarm, or trigger an interrupt to call some form of self-diagnosis program, which may attempt to actually locate the fault.

All circuits may be built using conventional methods and commercially available logic gates. The total cost of the hardware components is estimated at approximately $250.

GENERAL GUIDELINES

Small computers seldom have a large amount of core memory, often as little as 4,000 words. Often, external storage is limited to paper tape, which requires manual intervention to be read. In order for a real-time fault detection scheme to be useful in these circumstances, its memory requirements should be small enough to allow it to remain resident in a very small core memory and still leave room for the system programs.

A large number of faults are severe enough to either stop execution or at least to make sequential instruction execution impossible. No additional effort need be made to detect such a fault; the resulting failure to produce the required sequence of responses will cause the fault to be detected. However, faults which affect only a single bit position of a bus or register can be among the most troublesome. This type of fault may allow execution to continue, perhaps indefinitely, without an indication the fault exists.

Certain portions of the computer are very difficult to check under the assumed restrictions. Input-Output leads, other than those used by the monitor, cannot be checked without adding more hardware for each sėt of leads. Such Input-Output leads include Direct Memory Access, Data Channel, Console Controls, and special device channels. These devices usually appear as a data bus input which cannot be controlled. It can, however, be determined that none of these inputs is stuck at a one level as this would always cause the output of the bus to be one.

The instruction control circuitry cannot be checked directly. There are no instructions which access most of the control leads. Therefore, the correct operation of a control lead can only be determined by checking conditions caused by that lead, such as the complementing of a register. A thorough check of the control circuitry, under the given constraints, would be lengthy. It can, however, be determined that most of the gating signals can be produced. This can be done by executing every instruction type and checking to see that the proper data transfers have taken place. This method would intuitively seem to be very effective and can be accomplished in a short period of time.

Because the test routine is periodically required to change the contents of memory locations outside of its own boundaries, the interrupt facility must be disabled during the execution of the test. This is necessary to insure that all locations are restored before control is released by the test routine. Since it is undesirable to lock out high-priority tasks for a long period of time, the execution time of the routine should either be very short or the routine should be segmented to allow the servicing of high-priority tasks.

There are a number of ways to cause the external monitor to recognize that a fault has occurred. One of the most reliable ways is to design the test in such a way that the computer will output an unexpected bit configuration in the event of a fault. This technique is demonstrated by the following example. The accumulator is first loaded with a bit pattern A and then caused to skip over an instruction which would place A on the I/O bus. After the skip, the AC was changed to a pattern B and output to the monitor. If the skip did not occur, pattern A would have been output and would not have matched the expected pattern B.

Another method is to cause the program to "loop" in the event of an error. The normal sequence of output bit patterns will then cease and the monitor will recognize the cessation of response as an indication of a fault. Use of these techniques reduces the read-only memory requirements for the monitor.

TESTING TECHNIQUES

This section describes several techniques, which may be used to test the various computer components.

Registers

Registers in parallel synchronous machines commonly resemble the configuration in Figure 4. Although the actual circuitry may vary, the function is very

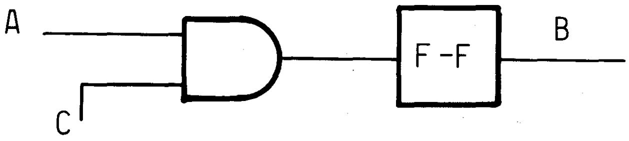


Figure 4—Flip-flop

simple and well defined. When lead C becomes enabled, the information on lead A (usually a major bus) is gated into the FF. When lead C is disabled, the FF retains the state of lead A. A variation of this type of register requires a fourth lead which clears the register prior to the enabling of lead C which then OR's the state of lead A into the FF. Verification that each bit of a register can be set to both logical states completes a functional test of the FF. A single possibility remains—a stuck at 1 fault on lead C. This fault is very severe because lead C is common to every bit of the register. Having lead C stuck at 1 would cause lead A to be fed directly into the FF and would totally incapacitate the register. In most cases the state of lead A changes many times between the setting and the reading of a register, thereby, destroying the original contents. If this does not happen automatically, an effort should be made to cause it to happen.

To check a programmable register, such as the accumulator, the register is first loaded with a bit configuration. The contents of the register are then output through the I/O bus to the external monitor. The programmable register is then loaded with the complement of the first bit configuration and its contents again output to the monitor. This procedure also checks the I/O bus drivers, and the outputs of each bus between the accumulator and the I/O bus.

Not all of the registers generally used in small machines are directly accessible under program control. In these cases a variety of techniques must be used in order to infer that the register can be set and read correctly. Five such registers were described in the introduction.

One of these registers is the arithmetic register (AR). Because the contents of the general purpose registers is commonly stored into AR so that AR may be used as the operand, many tests of the general purpose registers require the data flow to pass through AR, testing it at the same time. This commonly happens in the output instruction itself, meaning that no extra effort is required to check AR.

The second nonprogrammable register is the memory buffer (MB). This register is used to contain the data word core memory read-write operations; it must be capable of being set and cleared in order to correctly access core memory. If two complementary words can be read from memory, then the MB is operative.

The third nonprogrammable register is the instruction register (IR) which contains the OP code. This register is different from the above registers in that its contents are never gated to a point at which it could be displayed. In this case a set of instructions may be selected in such a way that together they incorporate both logical states for each bit. If it can be verified that each of these operations can be performed successfully, then it may be assumed that IR is operative.

The fourth nonprogrammable register is the memory address register (MA). This register is tested in the following manner: the contents of the sequence of addresses (0, 1, 2, 4, 8, . . . , 2^{n-1}, where n is the number of bits in MA, is saved in a test routine buffer area in real core for future restoration. Second, two complementary flags X and $\bar{X}$ are written into location 0 and read back. The flag X is left in location 0. Another flag (Y) is then written into each of the other n locations. If, after the n write operations, the contents of location 0 is still X, then the MA register is operative and the n+1 locations used for this test should be restored. To see this, assume that a bit (A) of MA is stuck at either 1 or 0. This is not to say that bit A is the only inoperative bit, but only one of possibly several. In this case, when an attempt is made to write into location 0, the flag will actually be written into another location (B). (If every faulty bit is stuck at 0, then B=0.) Later when an attempt is made to write into location 2^A, this data will also be written into location B, overwriting the original flag. In general, location B will be overwritten once for every inoperative bit in MA. When an attempt is made to read from location 0, the overwritten contents of location B will be returned, which will no longer be the flag originally placed there. It should also be noted that a fault in the address decoding circuitry or the memory itself cannot mask a fault in the MA.

The fifth nonprogrammable register, the program counter (PC), may be tested in much the same way as the MA register. The first flag to be written into location 0 is the binary configuration of a "JUMP TO A" instruction. Into the remaining n locations is written a "JUMP TO B" instruction. After the MA register has been checked, a transfer is made to location 0. This causes the PC to be loaded with a binary zero and the "JUMP TO A" instruction there to be executed. At location A, a signal is made to the monitoring unit that the transfer was successful and the contents of location 0 are changed to a "JUMP TO C" instruction. The code at location C will cause the program to loop, or output an unexpected word which will cause an error condition to arise. The test routine then proceeds to transfer to each of the locations 1, 2, 3 . . . 2^{n-1}. Each transfer should cause a return to location B, which simply continues the sequence. However, as with the MA register, if any bit of the PC is inoperative, then the "JUMP TO C" instruction will be executed and an error condition generated.

All of the above registers are found in most com-

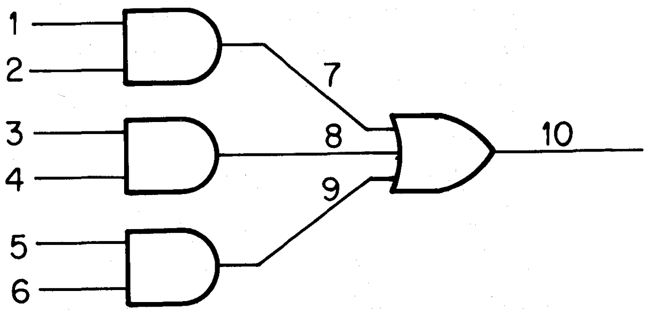


Figure 5—Bus structure

puters and the methods described are generally applicable.

In the methods for checking MA and PC, it is important to insure that addresses 0, 1, 2, . . . 2^{n-1} do not lie in critical parts of the test routine. This is simply a matter of convenience; if this restriction is undesirable, a slightly more general algorithm can be used to allow complete freedom in the location of the test routine. Since all of the n+1 locations lie in the first 2^n locations, the program could be loaded anywhere in the second half of the memory.

Data bus

A computer data bus typically consists of some variation of the circuit shown in Figure 5. Although there are many ways to implement this function, using different types of logic gates, nearly all faults will behave as though one of the leads numbered 1-10 has become frozen at one logical state. Certain varieties of logic allow the OR function to be accomplished simply by tying the outputs of gates A, B, and C together. This arrangement also satisfies the above statement. One input of each AND gate is usually common to all bit positions and functions as a gating lead for transferring data onto the bus.

A data bus may often be directly accessible to a number of peripheral devices for such features as direct memory access or a data channel. When this happens, one or more of the AND gates in Figure 5 may have inputs which cannot be controlled without interfering with certain of the peripheral devices. There will, therefore, be a few of the AND gates which will not be checked.

For the remaining AND gates, the objective is to verify that each input and output can assume either logical state. This can be done by gating both a 1 and a 0 onto the bus from every AND gate.

Control circuitry

Unlike a data bus or a register, the circuitry which controls the gating and timing for the instruction execution is neither simple in function nor standard in design.

The approach taken here is to include segments of code in the test routine which will exercise each instruction and check to see that it is performed correctly. This thoroughly tests the instruction decoder by providing it with all possible inputs. The control circuitry is the most difficult part of the computer to check because there is virtually no means of direct communication between this circuitry and the I/O bus. All tests must be made by performing an operation and verifying that the correct operations have taken place. It is possible, however, that something totally unexpected may happen in addition to normal operation of the instruction. To detect a fault such as this would require an extensive test for every instruction. There is no method known, at present, which would produce such a test using only the I/O bus with no hardware modifications to the computer. If such a method were available, it is likely that the storage requirement and execution time would limit its usefulness for a real-time environment. Fortunately, faults which occur in the control circuitry are usually quite severe and grossly affect the instruction execution. Therefore, a thorough exercising of each instruction appears to be the best approach to this problem and may be relied upon to detect the majority of such faults.

The length of the test can be reduced by studying logic diagrams in an attempt to find sections of logic used by more than one instruction. In some computers, for example, indirect addressing is handled by the same microinstruction cycle in every instruction. It would not be necessary, therefore, to test indirect addressing with each instruction.

Correct operation of certain instructions may be assumed if execution of the test routine is not possible without them. For example, it may be assumed that the JUMP instruction is operative because the first instruction executed after a hardware interrupt is a JUMP to the routine which is to handle the interrupt.

Although the approach used in this case is largely intuitive, if a little time is spent familiarizing oneself with the computer's logic and timing, the number of faults which can escape detection can be greatly reduced.

Logic function and condition logic

Most Boolean operations may be easily tested by exhaustion of their respective truth tables.

As a general rule, small computers have a ripple-carry adder. Because the circuitry is simpler, this form of adder is easier to check than one using carry-look-ahead. To check an adder, it is necessary to verify that each bit position can both generate and sink a carry. Both of these tests may be made simultaneously for all bits by adding a word of alternating ones and zeros to itself ($25252_8+25252_8$) and then adding that same word's complement to itself ($52525_8+52525_8$). It is also necessary to verify that each bit position can propagate a carry. This is accomplished by adding 1 to 77777_8. Adding $77777_8+77777_8$ verifies that every bit position can simultaneously generate and sink a carry. The remainder of the truth table may be verified by adding $00000_8+00000_8$, $77777_8+00000_8$, and $00000_8+77777_8$.

It is desirable to consult the logic drawings in order to determine how to test the overflow and conditional transfer logic. Although the implementation of this logic varies between machines, it is quite straightforward and simple to check.

Core memory

The core memory is, perhaps, the functional element which will fail most frequently. This may be attributed to the requirement for high-power circuits operating under closer tolerances than the conventional logic gates used in the rest of the computer. Many faults which occur in a core memory are the type which may be present for a long time before being discovered, such as a fault which affects only a rarely used part of memory. The ability to read and write at every address does not mean that the memory is free from faults; a faulty address decoder may read or write at two locations simultaneously.

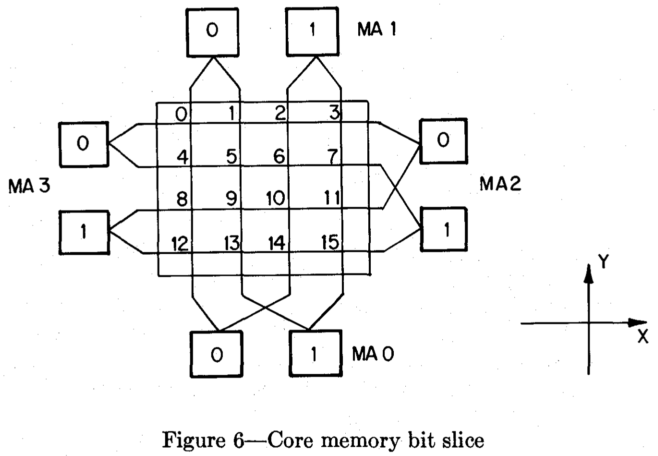


Figure 6—Core memory bit slice

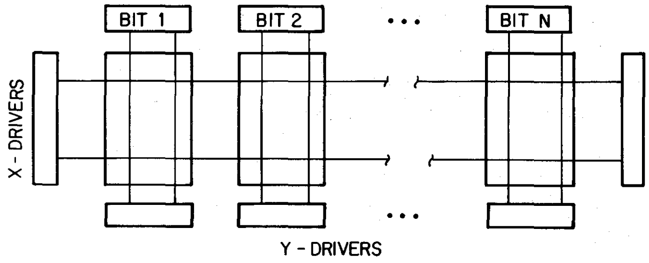


Figure 7—Memory addressing structure

Small computers commonly use a 2-1/2D arrangement for the memory address decoding.[8] This method relies on a coincidence of current on an X-axis lead and a Y-axis lead to select a core to be written or read. To do this the address bits are divided into two fields, which independently control the X and Y current drivers. In addition, each of these fields is further divided into two subfields. For example, if there are n bits in the field which selects the Y lead, then half of these bits will select one of $2^{n/2}$ current sources; the other half will select one of $2^{n/2}$ current sinks. Since every source is connected to every sink, the independent selection of one source and one sink selects one of the 2^n Y-axis leads. Different sources and sinks are used for the read and write operations.

Figure 6 depicts the addressing scheme for a 16-word memory requiring a 4-bit MA. Only a single bit position of the memory word is shown. Other bits of the same word are selected by replicating the Y-axis drivers for each bit position as shown in Figure 7.

During the write cycle, only the Y-axis drivers corresponding to bits which are to be set to 1 will be enabled. Because the read and write operations require currents in opposite directions, separate drivers are required for each of these operations. This means that a fault may affect the read operation and not the write operation and vice versa.

The test procedure for checking the address decoder is to select one of the four subfields, having length m, to be checked first. The contents of each of the 2^m locations obtained by selecting all combinations of these m bits and holding the remaining bits constant are stored in a buffer area. Each of the 2^m locations is then set to zero. Into one of the locations is written a word of all ones. Each of the remaining locations is checked to see that its contents is still zero. The original location is then checked and cleared. This process is repeated for each of the 2^m locations. Upon completion of the test, the contents of the locations are restored and the test is repeated for each of the remaining three subfields.

To justify this test, assume that one of the input leads to a read current source were stuck at 0. This would mean that at least one bit in the set of test words could not be read as a one and would be detected by the test. It is also possible that an input to a write current source could be stuck at 0. Because of a core memory's destructive read-out, the corresponding bit, or bits, could be read once and then never rewritten as a 1. Again, at least one bit of the set of test words would be stuck at 0 and the fault would be detected.

Suppose now that an input to a read current source were stuck at 1. This means that one of the 2^m addresses will select two read current sources at the same time, dividing the read current. The result will now depend upon the tolerances of the memory. The divided currents may not be sufficient to cause a core to switch. At least one test bit would then appear to be stuck at 0 and the fault would be detected. The divided currents may, however, each be sufficient to switch a core. During its execution, the test will cause a 1 to be written by using lead A. Later, when an attempt is made to read a 0 by using lead B, lead A will also be selected and a 1 will be read. Similarly, an input to a write current source may be stuck at 1. If the resulting split in the write current is insufficient to switch a core, the inability to write into certain test locations will be detected as before. If each half of the divided current is sufficient to switch a core, the writing of a 1 into some location C will also write a 1 into some location D. This extra 1 will be detected when location D is checked for a 0. Because this test is considerably longer than the previous tests, it may be desirable to partition the test. A very natural partition would be to check each group of current sources separately.

The only way to check individual cores in the memory is to read and write using every location. Although this must be a lengthy test, some time saving can be realized if the locations are checked by reading a word and writing its complement back into the same location. If the complement can be read back, the word is good. This method makes use of whatever is already in the memory location and, therefore, saves the time which would have been required to save the contents to the location and initialize the location. This test is normally not included in the test program because of the time required for its execution.

SUMMARY

This procedure has been applied to a DEC PDP-9 computer with two 8K core memory modules. The test routine requires 550 words of core memory and a maximum of 8 milliseconds per pass. The time needed to test the core memory increases with the size of the memory itself; but, by segmenting the test so that only a portion of the core is checked during each pass, it is possible to increase the memory size without increasing the amount of time required for a pass. All tests of the CPU itself are made each pass, but 12 passes are required to completely test the memory. The hardware monitor requires 58 words of read-only memory and solely determines the frequency at which the tests are made. This frequency may be adjusted according to the work load on the computer.

This technique would seem to have many applications on small machines which have previously avoided fault detection because of the cost or the need to make hardware changes to the computer.

ACKNOWLEDGMENTS

The authors would like to thank Messrs. Gary F. DePalma, G. Wayne Dietrich and J. S. Tang of Northwestern University for helping in the implementation of this fault detection scheme on the DEC PDP-9 computer.

REFERENCES

1 E G MANNING
On computer self-diagnosis: Part I—Experimental study of a processor
IEEE Trans Electronic Computers EC-15 pp 873-881 1966

2 E G MANNING
On computer self-diagnosis: Part II—Generalizations and design principles
IEEE Trans Electronic Computers EC-15 pp 882-890 1966

3 R W DOWNING J S NOWAK
L S TUOMENOKSA
No 1 ESS maintenance plan
Bell System Technical Journal Vol 43 pp 1961-2019 1964

4 K MALING E L ALLEN JR
A computer organization and programming system for automated maintenance
IEEE Trans Electronic Computers EC-12 pp 887-895 1963

5 C V RAVI
Fault location in memory systems by program
Proceedings of AFIPS Spring Joint Computer Conference
pp 393-401 1969

6 M S HOROVITZ
Automatic checkout of small computers
Proceedings of AFIPS Spring Joint Computer Conference
pp 359-365 1969

7 T R BASHKOW J FRIETS A KARSON
A programming system for detection and diagnosis of machine malfunctions
IEEE Trans Electronic Computers EC-12 pp 10-17 1963

8 P A HARDING M W ROLUND
Bit access problems in 2½D 2-wire memories
Proceedings of AFIPS Fall Joint Computer Conference
pp 353-362 1967

An organization for successful project management

by DON SMITH, JR.

Computer Sciences Corporation
El Segundo, California

INTRODUCTION

Successful software development requires that the product be accepted, that schedules are met, and that costs are acceptable. The number of software development efforts that do not meet one or more of these criteria is large. Conversely, there are many examples of successful development. The total number and variety of considerations that affect the probability of success is so large as to be beyond the scope of any individual paper. There is one important element of software development that appears to have been somewhat overlooked, however, and is the subject of this paper.

This paper presents the thesis that the probability of software development success is much increased by: (1) a proper separation of responsibility within the project organization, combined with; (2) extensive formal procedures; the combination of the two providing a system of checks and balances that gives continuous, accurate visibility of project status.

A general project organization is presented here that is felt to be widely applicable, and that achieves the required set of checks and balances. The formal procedures required to enforce the organizational division are not described completely or in detail, but examples are given.

It is felt that viable commercial products will only rarely be successfully developed with project organizations that do not provide some equivalent to the checks and balances approach described here. It can be argued that superior products may be developed at less cost and more quickly by small expert groups carrying the project from inception to completion. The technical contributions possible from such an approach cannot be denied, but the need still exists to place these contributions in an environment that assures their control. It should be noted that control need not be exercised as a function that unduly limits imagination. The intent of control is to assure that considerations of product function, cost and schedule remain clearly visible, and that decisions made which significantly affect such considerations are made consciously, with the effect of such decisions having received due consideration. Concerning the time at which control should be initiated, it is felt that "control begins when the first germ of an idea for a project appears in the organization—a discernible effort which will require an expenditure of organizational resources (human effort, time, money, physical resources) and lead to an organizational objective."[1] References 2 and 3 contain related reading not totally in accord with the above view.

There are, of course, many problems of software production other than those related to project management that can prevent the end product from being successful, such as: a faulty or unknown product definition; badly estimated product life; too rapid a development speed; major design problems; poor staffing; and inappropriate marketing (see also Sections III and IV of Reference 4). Good management can in many cases, however, assist in minimizing such problems by providing early visibility of the product and its problems, and by providing an environment allowing timely remedial action.

Prior to continuing with the body of this paper, one reference is particularly recommended for project planners. This reference, by Bemer,[5] contains an extremely comprehensive and useful "Checklist for Planning Software System Production." Reference 4, also by Bemer, is further recommended reading.

MAJOR PROBLEMS IN SOFTWARE DEVELOPMENT

This section describes the problem areas that are felt to be historically most common. The problems described in this section are felt to be significantly reduced by the organization and procedures later described.

In the most general sense, the two major problem classes are:

Those that make the customer unhappy; and
Those that make the developer unhappy.

Problems in the above two classes might also be classified as follows:

Unsatisfactory product
Schedule delays
Excessive costs

The above problem classes clearly are interactive, as schedule problems lead to cost problems, an unsatisfactory product leads to remedial action (cost, schedule), etc. Thus the division is to some degree artificial, but is useful for descriptive purposes.

The above three classes of problems will now be discussed in more detail.

Unsatisfactory Product

The major causes of user dissatisfaction with the product are:

1. Too Many Bugs
2. Instability
3. Unsatisfactory Performance

1. *Too Many Bugs*

The reasons for an excessively high number of bugs are many, including: (1) over-complex initial design; (2) implementation not consistent with initial design; (3) inappropriate implementation; and (4) uncontrolled evolution. These points are elaborated upon below.

The design may be so complex that the system in fact cannot be debugged. This complexity can be introduced in a number of ways. The initial design concept may, for example, unduly require complex module interactions, extensive time-dependent characteristics, multiple and non-obvious uses of the same information by many parts of the system, etc.

Complexity can be introduced during modifications following initial design if the initial implementors, or the next generation of designers and implementors, either cannot understand the initial design, cannot make it work, or the initial design is simply unsatisfactory in some way. In such instances an approach sometimes followed is to "program around the problem", leading to further problems in all areas of system performance, including possibly the undoing of some important initial design concept unknowingly.

A further cause of program complexity is due to the excessively "sophisticated," "elegant," "clever," or "efficient" programmer. Most, if not all, systems operating today have sections that either never will work quite properly or, if they do, dare not be modified to meet new requirements, due to their initial implementation by a programmer with one of the above attributes.

Poor implementation is a very common source of bugs. The cost of finding and repairing problems caused by poor implementation is so large as to make it quite clear that proper project staffing, detail design reviews, and strictly enforced unit test requirements are of extreme importance.

A situation leading to the existence of a continually high bug rate sometimes occurs when a product undergoes an evolution process in which new features are introduced simultaneously with ongoing maintenance. In such instances, synchronization between new features and bug fixes becomes a problem. An approach sometimes encountered under these conditions is to create a "safe" local environment by minimizing dependencies upon other parts of the system, even to the extent of creating duplicate or similar local tables, functions, etc. The problems in future maintenance and modifications are obvious.

2. *Instability*

The term "stability" will be used here to characterize the degree to which a job, once entered into the system, will terminate only due to a user error, and further, the effect of the user error is minimal (e.g., execution terminated, but files created to that time are not lost). If a user's program frequently must be rerun due to problems not of his own making, or if the severity of the problem is not consistent with the severity and probability of the error, then the system may be said to be "unstable."

Instability was not a problem in the days prior to multi-programming, as only one job was generally lost, and a rerun usually was successful. In multi-programming batch systems even extreme instability could be tolerated (and was) for a while, as a rerun often worked. In time, however, more capable I/O devices such as removable disks became available, with the attendant need for extensive support software. The major stability problem then became one of file creation and retention. This problem remains in many systems.

With the advent of real-time and time-sharing systems, stability has become much more important. In such systems there are, typically, from 20 to 100 users simultaneously somewhere in the processing cycle (in core with the CPU, in core, swapped, etc.). Losing the execution state, and possibly the files, of this many programs, or of critical programs, is extremely serious.

The most significant factors affecting a system's stability are the complexity of design and implementation, and the degree of attention paid by designers and implementors to automatic system recovery. Even good automatic recovery techniques cannot overcome problems inherent in an over-complex system.

3. *Unsatisfactory Performance*

The performance of a system may be unsatisfactory to the users. The performance of most batch systems is not so bad that users consider it intolerable, though it is still a major concern. The basic algorithms for obtaining satisfactory batch throughput are basically not complex, but as batch systems attempt to service an increasingly large community of hardware and users, the simplicity possible disappears. Thus, the major operating batch systems available vary widely in design, and in classes of service performed well. Batch systems with performance problems generally are troubled with excessive overhead, complexity due to the variety of user-types intended to service, difficulties in core management, and inadequate support for large data bases.

In the time-sharing area there have been notable instances where the performance was not what was required. A number of reasons exist for performance problems in time-sharing systems, for example, the basic assumptions were over-ambitious, not well thought out, or inadequately scoped (with modeling, for example). A more frequently encountered cause of unsatisfactory performance in time-sharing systems is the attempted retrofitting of a batch system to support time-sharing users.

Schedule delays

The most frequent cause of a missed schedule is, of course, the initial creation of an unrealistic schedule. Other reasons, generally related to the first at least implicitly, include ill-defined or missing requirements, changes in baseline assumptions, the customer interface relation, plus many of the causes of product and cost problems as described elsewhere in this section.

There are frequent instances in which the product was initially ill-defined and underwent a process of definition through its development. The conditions under which the initial estimates were given often change, including such changes in the available resources (money, skill, computer time). Requirements to support new hardware previously unenvisioned are often introduced.

The type of customer interface can have a major effect upon schedule. The customer interface is defined here as that person or persons who define, interpret, and control all requirements that come from the customer. The customer's relevant technical and management experience, maturity, willingness to respond to suggested changes, understanding of problems, and readiness to assist in achieving satisfactory mechanics of living together, are quite important. In particular, it is extremely important that the customer not require that all possible "bells and whistles" be included in his product. He should be reasonable in terms of tradeoffs, where considerable design and implementation simplicity can be gained by providing somewhat less function to the user. The customer must be convinced that simplified or limited functional capabilities will often improve the eventual performance and stability, and will increase the ability to add future enhancements to the system.

Excessive costs

A major software development problem for the developer, in addition to user-related problems, is cost. Major causes of cost over-run include:

1. Schedule Delays
2. Low Initial Estimates
3. Staffing Too Rapidly
4. Staffing With Quantity, Rather Than Quality
5. Follow-on Costs
6. Type of Contract

1. *Schedule Delays*

Delayed schedules generally lead to higher cost, unless the developer had the foresight to see the schedule was unrealistic and staffed according to a reasonable schedule.

2. *Low Initial Estimates*

Some excellent techniques for estimating project schedules and costs have been presented in the literature.[6] Persons making schedule and cost estimates must previously have been closely involved with projects of similar size and complexity. They must have accurate cost and progress summaries of the previous projects. It is useful to know the original estimates for previous projects, and eventual results.

The value of PERT-type techniques in making initial estimates is limited, due to the lack of detail concerning the project available at that time. A PERT network or the equivalent should be constructed, however, even if only on a very gross scale, as a means for forcing upon the estimators a mildly rigorous framework in which to place their estimates. Of particular importance in the PERT are, of course, those things that frequently lead to schedule problems. Foremost among these are a knowledge of: (1) dependencies outside the project; (2) the period required for initial staffing, defining the organization, and other activities required to have the people know what they are supposed to be doing, why, and when; and (3) those major activities within the project which may proceed in parallel, and which must proceed serially.

Machine time estimates are almost always underestimated, on projects of all sizes, and usually by a large factor (two to four). Even when machine time estimates are made by reasonable and experienced people, they are often made invalid by the eventual uncontrolled and wasteful manner in which the computer is used. Even today we find use of "block" or "hands-on" time on large, fast machines. Unless projects develop techniques for optimal use of computer time, including the support software required to remove the need (except in rare cases) for hands-on time, problems of cost, geographically separate computer installations, and inability to maintain schedules will generally arise.

Some activities in project development can proceed quite nicely in parallel by defining the major interfaces early. For example, a compiler can often be developed in parallel with an operating system, as its interface needs are small. Similarly, applications design and coding may often overlap development of the support system.

3. *Staffing Too Rapidly*

Staffing a project is an important and complex issue. There are never as many good people available as a manager would like. A manager is aware that a certain general level of manpower will be required to do a job, and then the staffing search begins, trying to identify people who are available, have relevant experience and skills, etc. Initial estimates that a project will need a certain number of programmers should not lead to the addition of "head count" simply to meet predictions. In general, projects staff too quickly with people anxious to produce code, and not quickly enough nor with enough skill with managers and supporting staffs (e.g., integration and test activities, described later). Programmers are often brought on board in order to start doing something before they really know what to do. A good generalization is to always have the minimum number of programmers possible on a project.

4. *Staffing With Quantity, Rather Than Quality*

No matter how insensitive, trivial, simple, or straightforward a piece of code is to write, it will cost more, often much more, if a poor programmer is given the job. Sometimes the cost is hidden, perhaps for years, but once programs "work" in some sense there is a great reluctance to replace them, which tends to extend the life of some rather poor examples of the programmer's art.

In such complex areas as systems or real time software, unless there is a very large amount of review talent available, people should not be brought onto the project unless they are extremely skilled. (It can be argued that this statement should be enlarged to include all types of programming.)

Relevant experience and resumes of people cannot be considered the most important criteria for hiring. Extensive experience cannot overcome the lack of an organized approach, insight, maturity in tradeoffs, and a willingness to make very sure that a program really works correctly.

It should not be assumed that key people will be available through hiring, or transfer from other parts of the company.

5. *Follow-on Costs*

Unrealistic estimates of follow-on costs after the first release of a system are often made. Follow-on costs may include maintenance, enhancements due to oversights in the initial product (frequently stability and performance problems fall into this cateogry), user enhancements required to stay competitive, a need to support new hardware, etc. The larger a software system is the more reluctance there is to replace it. Thus the product life may be long, with a fairly continuous, and costly, development cycle.

6. *Type of Contract*

The basic types of contracts are cost-plus fee (percentage of cost, award, incentive, etc.) and fixed price. Virtually any type of contract is satisfactory if the contract administrators on both sides are both knowledgeable and reasonable. Cost-plus contracts are felt to be superior to fixed price contracts for all cases except the (relatively rare) instances in which the product, dependencies, etc., actually *can* be specified in great detail. For example, the author has observed in some detail a cost-plus contract with highly skilled technical and management interfaces, and a fixed price contract with a vague contract and inexperienced interfaces. The cost-plus contract could have been costly, but was not, due to the expertise of the contract managers. The fixed price contract in theory limited the cost, but in fact, due to extensive requirements above the scope of the contract, resulted in contract renegotiation, higher cost, schedule problems, ill will, and eventual replacement of contract managers on both sides.

There are, of course, instances of cost or product problems in cost-plus contracts, if, for example, expensive and not necessarily talented personnel are "dumped" onto the contract for profit and/or convenience. A sometimes-encountered problem in fixed price contracts is a "buy-in," followed by either minimum possible contract interpretation or change of scope. Again, experience and reasonableness on both sides is required in order to prevent the contract type from perhaps heavily contributing to cost and other problems.

OPTIMAL PROJECT ORGANIZATION

This section describes a general project organization that assists in providing a check-and-balance relation in a project that will contribute to project visibility, control, and other factors important to meeting project goals. This organization, and the corresponding procedures of which examples are given, is felt to have wide applicability, but is not, of course, an exact model of any particular project. An organization will inevitably be shaped by the job to be done, people available, company policies, etc.

This section discusses the following topics:

General Project Organization
Project Managers
Development Activities
Integration Activities
Test Activities
Procedures
Staff and General Support Activities

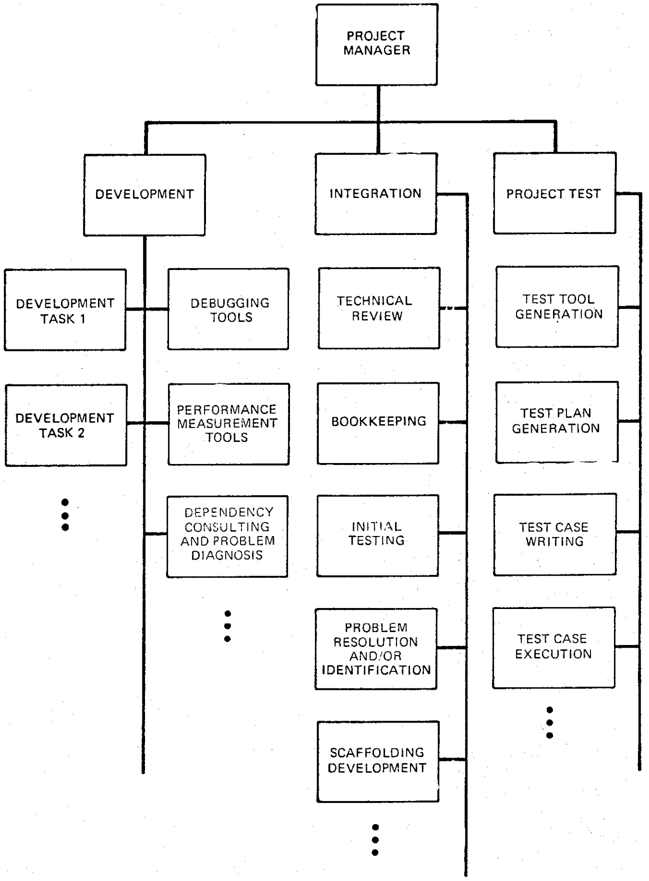


Figure 1—Project organization

General project organization

Figure 1 shows a project organization that it is felt is generally applicable for a wide range of development projects. Many of the functions shown in this figure are not, of course, required precisely as shown in particular projects, but the major functions implied, or their equivalent, are felt to be required in some form for projects on the order of five people or more.

The key element of the figure, and indeed of this paper, is the division of responsibility into separate functional groups, such as those termed here Development, Integration, and Project Test. This division is felt to be an absolute necessity. This division must occur early enough in the project life so that each functional group can adequately prepare.

The following sections describe the Development, Integration, and Project Test groups in more detail. The general responsibilities of these three groups are:

Development:	Design, write, debug and unit test new code.
Integration:	Integrate all code. Be the project "bookkeeper."
Project Test:	Determine the status of integrated code.

It is critical that people who are developing the product not be simultaneously responsible for all the bookkeeping functions, integration of new code, and confirmation that the integrated code works. The project organization must be such that all groups have strong technical people, and receive full support from the project manager. Without an appropriate division of responsibilities, and a corresponding set of procedures, both visibility of true project status and control over development are very difficult to achieve. Among the typical problems that result are: not knowing what code is really in a particular version of the system; and a requirement to accept a programmer's statement for the status of a module ("90 percent done, only a few minor things don't work").

Project managers

The three levels of project management discussed here are:

1. Project Manager
2. The Project Manager's Manager
3. First Level of Management Within the Project

One comment relative to all levels of management should be made before going into further details. Managers are assigned a responsibility, and must be allowed to make decisions within their assigned responsibility without undue requirements to justify their decisions. If a reasonable set of checks and balances exists (e.g., Review Boards), and if the managers chosen have appropriate experience and judgment, then their scope of decision making should not be frequently tampered with. Frequently no decision is worse than nearly any timely and moderately rational decision.

1. *Project Manager*

The most important single person on a project is the project manager. Therefore, choice of this person should be made with appropriate care. There are a wide variety of attributes that it is desirable to have in a project manager, and a set of attributes that if they exist in a project manager almost assuredly doom the project to failure.

The project manager must be technically strong enough to understand the important technical decisions made by those under him, to ask the right questions about those decisions, and to change them if needed. In particular, he must be able to cause the project to move toward realizable goals, and be able to say "no" to impractical ideas, and enforce it.

The project manager must have both technical and managerial maturity. All too often managers are chosen on the basis of strong technical skills plus a desire to be "promoted." Opposite extremes would be the assignment of a weak technical person to a managers role because nobody else would put up with the customer interface or personnel problems, or because the person just happened to be available.

Another characteristic that must be contained within the project manager is a desire to become *involved.* He must *manage* the project, not simply spend occasional time in reviews, believe all that is told him, and dilute his activities with marketing for new business or other company activities. It is always easier for a manager to assume that since good people are working in a certain area, and everyone is working hard, the end result will work out all right, and that he need not take the time to dig into that area in detail. On reflection, it almost always turns out that one or more areas do not develop satisfactorily, regardless of the skill of those most closely involved. Frequent reviews of design, implementation, test status, dependency status, schedule milestones, etc., by the project manager, are required.

It is not felt to be possible for a project manager to be in some sense a "pure" manager, that is, manage PERT charts, organization charts, staffing, facilities, and so forth, and allow others to make the larger technical decisions uninspected. This would rule out software project managers whose experience is only in other types of work.

2. *The Project Manager's Manager*

In addition to the project manager being of appropriate technical and supervisory capability, it is also critical that his immediate supervisor have relevant experience, a set of both common and complementary skills, and not be so sure of the project manager that an independent check of status and problems is never made. Every project manager needs someone reviewing his major decisions, a person to whom he may bring problems and receive real help, and someone to just talk to at times in order to remain objective about his project.

3. *First Level of Management Within the Project*

A superior project manager can, with appropriate organization and procedures, control up to 20–30 people satisfactorily with a set of managers reporting to him that are really "technical task leaders"—strong technical people with some supervisory capability, each supervising 3-5 people. Above the 20-30 level, however, support of full-time supervisory people within the project is required.

Development activities

Figure 1 shows a number of possible functions to be included within development. Many additional or different functions exist, of course, for any particular project. This is felt to be a reasonable example set, however. The particular activities to be described in this section are:

1. Major Development Function
2. Debugging Tools
3. Performance Measurement Tools
4. Dependency Consulting and Problem Diagnosis

Before beginning a discussion of the above, it should be mentioned that this paper does not attempt to discuss the merits of such very important development considerations as: software specification or implementation languages; types of debug tools; tradeoffs between speed, space, portability, flexibility, cost and schedule; the value of formal proofs; etc. The emphasis here is on functions that must somewhere be performed, not how they are best performed.

Two characteristics of the development process that frequently occur will first be briefly mentioned: the effect of design changes during implementation; and the occasional need to replace significant design or implementation efforts. A great deal of interpretation of the initial high level design takes place during detailed design and implementation. If not carefully controlled (via documentation and review) it is possible to develop a product substantially different from and/or inferior to that initially intended.

Occasionally it is realized that a significant mistake was made in design or implementation. The temptation always exists to "live with it," "fix it up later," or "program around it." Generally such approaches are a mistake—significant problems must be removed when discovered, or at least a plan for their removal created and begun. Too often the problems, if put off, remain in the system for a long period of time and cause far more difficulty than if remedied early.

1. *Major Development Function*

The major development function is, of course, responsibility for the overall design, detail design, debug, checkout, and internal documentation activities required for product implementation. Worth mentioning is the undesirability of fragmenting the responsibility for this effort in a manner that results in a manager with overall knowledge of the entire development not having full control over work for which he is responsible.

2. *Debugging Tools*

The creation of debugging tools is a distinct programming task, with substantial requirements to be defined, lead time needed for development, etc. In general there remains far too much dependence upon octal or hexadecimal dumps as the major aid to resolving difficult problems. Including appropriate (e.g., brief, symbolic) tools in a system initially is generally inexpensive; adding them later is quite difficult.

3. *Performance Measurement Tools*

Performance measurement tools must be included in development planning. This requirement needs planning, lead time, etc. The capture and processing of performance information is something that should go on very early in a system, but often does not. Obtaining early indication of performance problems is often critical, however, as their repair can require major surgery.

4. *Dependency Consulting and Problem Diagnosis*

Programming projects usually have significant dependencies on areas outside their direct control, for example, the hardware employed, support software, and communications facilities. Where such dependencies exist, it is best to assume that non-trivial problems will occur, and place in the project appropriate skills to isolate the problems and assist the appropriate support group in their removal. Since such problems often arise due to misunderstandings resulting from inadequate documentation, or perhaps just a complex dependency, the ready availability of consulting assistance within the project will be valuable.

Integration activities

Put briefly, this group's activities are to provide a major portion of the independent *control* and *visibility* of development activities that is required. The Integration Group lies organizationally between the Development Group and the Project Test Group. This group receives new code from Development; integrates the various pieces of new code received from Development; performs the bookkeeping tasks on detailed status of all changes that have come in; assures proper approval levels; and other similar tasks, prior to passing on a new release for testing.

The *control* over development resulting from this sort of activity is obvious—nothing goes into the system unless properly identified and approved, and extensive documentation exists should the changes need to be altered (or, in extreme cases, removed). The *visibility* provided above an approach which has Development personnel alone report on system status is a bit less obvious, but critical. Development personnel tend to be optimistic about their code, tend to report a feature as fully provided even though only part of it is done, and to otherwise exhibit human traits. Integrations' job is to be more hard-headed—something is done only if the code was turned in, with appropriate documentation (e.g., updated internals), and a repeatable test sequence (without which the change is not accepted). Thus a project manager can know *exactly* what is completed.

It is becoming more and more common in the industry for the integration function to include very strong technical people, for such purposes as: (1) providing the ability to inspect new code submitted to the system for general technical correctness; and (2) identifying integration problems well enough to fix some, and if not to provide good information to Development people. If all the the project technical strength lies in other groups, the Integration Group must always borrow technical resources from these other groups, which they give up grudgingly. An intermediate approach is to assign people from Development to each new release, thus providing additional technical muscle for integration activities.

Another important function of the Integration Group is to perform some level of initial testing on the product in order to assure that the major new functions will work together and were integrated properly. Typically a small set of regression tests is run by Integration to assure that no major regressions have occurred prior to approving a new system for more extensive testing.

In addition to the above principal activities, a number of other activities may be convenient to include in the Integration Group. An item that may come under Integration is one that is termed here "scaffolding development." In the development of a new project it is always required that some subset of the system be generated first as an anchor on which the fuller system can be developed, that some other system be used as a vehicle for creating assemblies, load elements, etc. The term "scaffolding" is used to define the set of code and procedures that is developed to substitute for missing interfaces or missing subsystems in order that a partial system can run, or that phasing from one system to another goes smoothly.

Enforcement of such standards or requirements as linkage and naming conventions and memory utilization may also be conveniently performed by Integration. Automated tools may help such activities.

Test activities

This section describes a series of testing levels, and their place in the project organization. A fundamental corollary of this section is the now (hopefully) clearly established requirement that successful projects must include: (1) early creation of test plans, procedures, and staff; and (2) one or more groups established for testing a product that are organizationally distinct from and equal to the Development Group.

A testing approach that has proved quite successful is one in which there are a number of separate and distinct test activities, with minimal overlap, each fulfilling a needed function. A possible set of testing levels, chosen for purpose of example, is the following five level set:

1. *Unit Testing* by Development programmers. These tests show that the new function(s) operate in a private environment. Development managers must review the unit test plans in detail, as a poor job at this level will be very costly later.
2. *Integration Tests,* including rerunning of some of the above unit tests, testing the ability of the system to operate without major regression such that the next testing phase can concentrate on testing new code.

3. *Project Test.* This is the major test activity and is discussed in more detail below. This group runs extensive tests on new systems to confirm that new features work.
4. *Regression Test.* In addition to testing that new features work, one must also ensure that things that used to work still do. This may be done either within the project (e.g., by Project Test), or by an organizationally separate group. The regression testing activity differs somewhat from Project Test activities in that: (1) the volume of testing is larger; (2) the features tested are generally more stable than the new features; and (3) the regression testing group has more lead time, better documentation, etc., than the group testing new features, thus need not be quite as good "detectives" to determine what the new features really are.
5. *Field Test.* Regardless of the amount of prior test, non-trivial bugs always exist in the final release. For this reason a field test should be scheduled, in which the new system is mounted in one, or a few sites, and watched for a while, prior to full release.

The Project Test Group normally performs the following functions:

1. Test Tool Generation
2. Test Plan Generation
3. Test Case Writing
4. Test Execution

1. *Test Tool Generation*

Test tool generation requires both "hooks" in the system, and programs to process and analyze the results. To minimize the requirements for support upon the Development Group, it is advisable to develop test tools that are carefully designed to capture all information that is likely to be useful, but impose minimum space and complexity requirements upon the system. These tools should not be the last functions integrated. With early data definition, the required data processing programs may be developed in parallel with the system. Typical of the test tools that are required for development of modern systems are a means to easily and repeatedly run large series of programs. For example, a common practice is to simulate a large number of users in a time-sharing environment, with input coming from precatalogued files rather than on-line users, but seen by the system in a very similar way. These "prestored" jobs may contain test programs, a sample of real user programs, or both. Information captured from these test tools should record such information as is later needed to determine such things as response times, if or not the output compares on a line-by-line basis to previous runs, and so forth.

2. *Test Plans*

Extensive study of documentation and listings plus detailed planning is required to develop a set of test plans that may be reasonably executed and analyzed. Test plans must include enough detail so test cases are not written with dependencies on functions not yet integrated. Among the many types of tests that must exist are those for testing: functional capabilities; stability; performance; fallback (the ability for a system to be replaced by an earlier release if it fails); and the ability to support various hardware variations, including degraded mode operations in which only a subset of the hardware is available.

3. *Test Case Generation*

This is a difficult effort to do well if the test programs themselves are to: (1) not produce more bugs than they find; (2) efficiently and completely implement the test specifications; and (3) be self checking to a large degree.

4. *Test Execution*

Test execution includes the actual running of the tests, creation of files used for later analysis to determine if the tests ran properly, documentation of results, getting the information back to the Development Group and consulting on procedures used, and so forth. With automated test tools this effort can be reduced very significantly, with only a few people required to test features not easily tested automatically.

Procedures

The type of procedures required to assist in providing project control and visibility may be very large. This paper does not attempt to give a detailed set of procedures. The following list, provided as an example, gives information which would be expected to be submitted by all Development programmers when their code is ready for Integration.

System level upon which the changes are based
System level intended for inclusion of the changes
Change identification
Change description
Listing of the changes
Card deck (or the equivalent)
Changed internal documentation
Changed operational documentation
Changed external documentation
Dependencies upon other changes
Modules changed
Copy of the unit test passed to confirm unit testing, and instructions for reproducing the test results in the integrated system
Approval signatures

For an example of support software to aid in implementing procedures of the above type, see the CLEAR system description of Reference 7.

A further example of a procedure usually needed is an automated planning aid of some sort. There are a number of reasons why a mechanized aid (such as PERT) is useful. These include:

1. A common basis for discussing project status is defined. Even if it is not quite everything that would be desired, everybody gets used to using it, and the shortcomings are overcome.
2. The mechanical difficulty of experimenting with changes in work required, people available, slipping dependencies, etc., is much less, as one need change merely a few data cards and rerun a program to get an initial feel for problems, tradeoffs, etc.

To meet the above objectives, the scheduling aid must have a number of attributes, including: (1) the output must be prepared by computer, not the local art department, as waiting a week to see the latest chart makes the whole thing useless; (2) the input preparation must be relatively simple; and (3) the process must be flexible enough and closely enough related to the projects' detailed activities to be a widely usable day-to-day planning tool (for example, listings should be produced that sort activities by start date, end date, person assigned, release version, etc.).

Staff and general support activities

There are a variety of activities that must exist within either the project or a supporting organization that do not always fall neatly into the Development, Integration or Project Test Groups. Those discussed here are:

1. Staffing Assistance
2. Customer Interface and Support
3. Review Board
4. Operations Support
5. Documentation
6. Status, Cost and Schedule Data Collection, Reporting

1. *Staffing Assistance*

In the early days of a project, when many people must be interviewed, resumes must be reviewed, and so forth, the project manager should not have to (and cannot) do all the recruiting for the project. It is also felt, however, that this job cannot be delegated completely to a personnel department. A good personnel department can be very valuable by performing such activities as obtaining candidate names and performing initial screening for quality and relevant experience. Technical men on the project must do all detailed screening, recruiting and staffing for the project, however. This is a difficult job to do well, as people range from uncommunicative geniuses to imposing charlatans. Nevertheless, detailed interviews combined with extensive checkups of references can assist greatly in minimizing the dead wood brought aboard. Realistically, it must be faced that more people must be hired than are needed, as some will not work out and must be discarded in short order.

2. *Customer Interface and Support*

A major element of the customer interface must, of course, be the project manager himself. This is so since he alone is responsible for the project, and therefore only he (or his manager, but hopefully not independently) can commit the project. It is often the case, however, that there is a need for frequent discussions between various people within the project and within the marketing organization, or other organizations working with the project, such that a continuous and heavy transmission of information back and forth across the project interface must exist. The project manager is generally too busy to both perform this function and manage the project, so it is wise to set up a small group, perhaps one person, reporting to the project manager, to provide a continually available interface to the customer. This may include such customer support functions as assisting in preparation of rough external documentation and/or proposals for the customer, looking in advance at conversion problems when the system becomes operational, etc.

A further important role of the customer interface may be one of providing appropriate visibility concerning progress and problems. It is fairly easy for a project manager and a customer to both be working hard and think the other one knows what each is doing, but to discover that in fact major differences of opinion occur. A good interface can reduce these problems.

3. *Review Board*

A review board should be established that includes people who are generally not assigned full time to the project, but remain cognizant of technical activities going on in the project through their continuous participation on the board. This board includes senior technical people from wherever is appropriate. The purpose of this review board is to review major decisions either already made or that must be made before going on. This helps assure that all relevant experience available is brought to bear. Items reviewed should be of both technical and administrative (e.g., cost, schedule) nature.

4. *Operations Support*

Computer operators need someone to call on for help occasionally, efficient machine room procedures need to be developed that are generally rather different in project development from those in a typical production shop, etc.

5. *Documentation*

Documentation is always a problem, both where to place it organizationally as well as who should do it. There is a very wide variety of types of documentation to be done, usually far more than is obvious at budget time. Among the major types are, of course, the internal documentation for the system (usually done in Development, with the support from elsewhere). Another obvious type is the external documentation for the user. There is also the question of documentation to support people who must run the computer center, documentation for user managers, training documentation, etc.

One of the major unsolved problems of the computer industry is the creation of appropriate documentation, at a reasonable cost. It always seems to get done eventually, but is always painful to the programming staff, and not infrequently to users.

6. *Status, Cost and Schedule Data Collection, Reporting*

Projects sometimes begin with good procedures in place for creation of status, cost and schedule collection, etc., but as time goes on less and less use is made of such procedures. A frequent cause is the lack of staff to regularly collect and report upon the data collected. This can be a substantial job, but if done properly provides management with early indications of potential trouble areas.

SUMMARY

The proper management of programming projects requires means for visibility and control. To know what is actually going on in a project, to not have critical path activities slip, to head off such difficult-to-solve problems as memory over-use and slow performance, and with all this to meet a tight schedule and budget, is a challenge met by few. The primary attributes that must be contained in a project to provide good visibility and control are:

1. Division of responsibility in a manner that gives good checks and balances (separate groups for Development, Integration and Test).
2. A large set of procedures, rigidly enforced, that supplement the above organization by forcing an automatic flow of information.
3. Managers, supervisors, and task leaders at all levels that have good technical judgment, and will "dig" to find out the true status of all work.

REFERENCES

1 C KING
Systems analysis and project management
McGraw-Hill Book Company New York 1968 p 255

2 M E CONWAY
Datamation Vol 14 No 4 April 1968 pp 28-32

3 R H KAY
The management and organization of large scale software development projects
AFIPS Conference Proceedings Volume 34 1969 SJCC pp 425-433

4 R W BEMER
Manageable software engineering
Software Engineering Volume 1 Computer and Information Sciences Academic Press 1970

5 R W BEMER
Checklist for planning software system production
NATO Conference Software Engineering Report on Conference 1968 pp 165-180

6 J E ARON
Estimating resources for large programming systems
NATO Conference Software Engineering Report on Conference 1969 pp 68-79

7 H M BROWN
Support software for large systems
NATO Conference Engineering Report on Conference 1969 pp 53-60

Commercial data processing machines in government applications

by J. D. ARON

IBM Federal Systems Division
Gaithersburg, Maryland

INTRODUCTION

When a major military weapons system is designed, it is often so far ahead of the then current commercial technology that special approaches are needed to make the system feasible. This has led to the development of many special purpose computing systems as components of large government procurements. But special purpose hardware is often too complex to be built on normal schedules; furthermore, the software to go with it must be deferred until the special machine is finished. The sequential nature of the hardware/software development cycle means that any slip in the schedule of either element directly affects the total system schedule. One obvious way to avoid cost and schedule problems due to spec:al purpose computing systems is to substitute commercially available comput ng systems for them.

There is a lag of some 3-5 years between the specification of a major government application and the detailed design of related non-government systems. Dur'ng this period, the hardware and software designs of the computing :ndustry generally incorporate the special characteristics of the government system for their standard product line. By the time the government system goes into operation, contemporary commercial systems have caught up. Thus, the convergence of government and commerc'al needs is leading to standard computing systems which meet all but a few of the needs of the special military and federal civilian applications. *In those cases where the commercial system is totally responsive to the requirements of the special system, it offers sufficient benefits to recommend it in preference to special purpose hardware.* In other cases where the commercial offerings are only partly responsive, there are still significant advantages to the use of commercial systems (modified as necessary) to be evaluated by the system manager.

BENEFITS OF USING COMMERCIAL HARDWARE

Salient benefits

The benefits of using commercial hardware depend on which elements of the system design hav e the highest priority in the mind of the user. For instance, in evaluating commercial data processing alternatives for the SAFEGUARD system, the Army l:sted five system characteristics in relative order of importance and weighted them to emphasize the ranking. Performance, including throughput and some aspects of reliability, carried 40 percent of the weight. Product Deliverability was next in importance, followed closely by Implementation Risk. Some 10 percent of the weight was assigned to Growth Potential and Cost carried a nominal 5 percent. Commercial alternatives will score very high in the last four characteristics. Will they stand up in Performance? Probably, yes. The computing performance of general purpose mach'nes has been growing by an order of magnitude every five years. There is no sign that the growth will stop.

The commercial computer approach is strongly enhanced by the availability of compatible families. This feature of the commercial market was introduced in a limited way in 1962. Taking advantage of this, for instance, the USAF Command Post Project 473L was able to install an IBM 1401 computer very early in the system design stage to test out concepts related to disk storage and display consoles. Later, as the workload on this facility grew, the 1401 was replaced by the larger 1410 without substantially altering the system. Through the emulation capabilities of the third generation S/360, some of the original 473L programs could still be used after the 1410 reached the limit of its utility. Most manufacturers now offer commercial families of compatible machines. The same degree of compatibility is seldom offered in the special purpose computer market for simple economic reasons: neither the buyer nor the vendor has a cost justification for developing any features other than those directly applicable to the special application. Consequently, there is less growth capability in special purpose machines procured for military systems than there is in the standard commercial product line of a given manufacturer.

Referring again to 473L, the use of the 1401 as an interim test bed illustrates a common aspect of the system development process. In order to make progress on the overall system it is important to overlap many activities. Programming, as an example, will proceed while the hardware is still being built. In 473L it was necessary to create a simulator of the Librascope L-3055 computer, specially procured for the project. The simulator ran on a commercial 709 located in the programming facility. By using the simulator, the programmers were shielded from delivery delays in the L-3055.

The rigid product test protocols observed by most manufacturers further increase the confidence that commercial machines will perform according to their advertised parameters. The risk, as has occurred on special procurements, that a computer will be delivered with some of its instruction set still undefined and perhaps not included in the circuitry is negligible with commerical machines.

If the performance and reliability of commercial machines are sufficient to justify their use to the systems engineers who design the military and federal civilian systems, then the software support available with the machines should throw the decision well over in favor of the commercial product. *Software, which includes development aids as well as the deliverable computer programs, has become the most difficult aspect of large systems to plan and control.* Project management is generally unfamiliar with the nature of software and lacks the experience to estimate and schedule the programming effort accurately. However, since a data processing system is a package of hardware and software, in which the software is the only feature directly accessible to the user, the system cannot be completed until the software has satisfactorily passed its tests. It is specifically in the area of software that commercial machines have a marked advantage over special purpose implementations.

Vendor software, available with the first delivery of machines, normally includes a basic operating system which controls the hardware function and facilitates the preparation and execution of user programs. In addition, the vendor supplies a variety of language processors: basic assembly language for manipulating the hardware instruction set, higher level languages such as FORTRAN, Algol, COBOL, JOVIAL, and PL/1 for application programs, and simulation languages such GPSS and SIMSCRIPT for building models. The higher level languages are easier to use than the basic assembly language and speed up the development effort. The languages have the added benefit that they are easy to learn and can be transferred from one machine to another with relative ease.

The advantages of higher level languages are so self-evident that some have been written to run on special purpose processors. But the resources applied by the general purpose machine manufacturer to come up with a comprehensive efficient high language processor are too extensive to be duplicated on each special processor. The result is most often a limited subset of the full language and an inefficient compiler.

This last point, when extended to cover additional areas of vendor support and product enhancement, emphasizes one more significant benefit of commercial machines: *Commercial systems are usually less expensive because they achieve economies of scale inaccessible to special products.*

Characteristics of applications of interest

There are three general classes of government applications which can be identified according to the stringency of their unique requirements:

Class I—Problems such as inventory management, management information systems, finance and accounting, etc., which are performed daily or less often in an office environment.

Class II—Problems such as assisting commanders in operational situations or controlling weapons and space missions. An office environment with special security, reliability, and availability provisions is required in fixed installations. Installations in aircraft or ships may call for special equipment.

Class III—Problems such as air navigation, weapons guidance, or space vehicle maneuver control in an environment in which temperatures and other physical parameters can vary widely and where the computing element may have to be shaped so as to fit into a small space. Reliability must be very high and maintenance may be impractical once the mission has started.

Class I problems are normally handled by commercial hardware. Class II problems are dominated by the environmental constraints and are normally handled by special hardware*. Class II problems, which are the primary subject of this paper, present the opportunity for tradeoffs between special and standard hardware. The typical tradeoff to be made is whether to place a specially designed machine in an uncontrolled field environment or to create a spec'ally designed controlled environment in which to place a commercial machine. The commercial machine approach is most appropriate for fixed installations at headquarters locations and for transportable facilities in vehicular vans and airlift pods. The decision can swing either way for ship and submar:ne installations. The airborne case will almost always call for special hardware to meet space and weight constraints.

The SAFEGUARD advanced ballistic missile defense system falls in the second class. So do Air Traffic Control and the ground support for the Apollo manned space flight system. In each case there is a need for large capacity storage, fast computation, versatile input/output devices, communication with other systems, high availability, and the flexibility to respond to changing workload requirements.

Performance

No effective measure of computer performance has been developed that applies to more than one situation; therefore, it is common to describe performance requirements by the arbitrary value of the instruction execution time. The lack of other throughput requirements is usually due to the inability of the system designers to anticipate the effect on throughput of interactions among the programs in the job stream. At best, the designers can lay down specifications for system throughput based on the real time requirements of one critical activity. Most of the complex systems of interest to the government involve many related activities; e.g., the relations between radar data and aircraft tracks and between aircraft tracks and flight plans and between aircraft tracks and traffic control procedures. The relationships cannot be unravelled so that the system design requires all of the computing to be done on a single computer configuration† where all the related information is available at the same time. While it may be useful to categorize machines based on a single parameter, selection of a machine for a particular application should be based on a more thorough analysis of its capability to handle the specifics of the job. The selection of computer hardware should be one of the last things done in the design phase. Premature selection will force the designers to distort the rest of the system to conform to the computer. Problem definition, workload sizing, allowable computing overlaps, deferrable computations, absolute real-time deadlines, I/O rates, and other job descriptors should be determined first. Then a balanced computer configuration can be selected (using a gross model for timing and storage evaluation) which will do the job. If this policy is followed, the lead time for computer delivery will be compressed.

There are very few performance requirements that cannot be handled by off-the-shelf hardware. The performance criteria in which commercial machines exceed the capabilities of most special purpose machines include:

1. Range of speeds available for a family of central processors;
2. Range of memory capacity for a given processor or family of processors;
3. Number and diversity of input/output devices available.
   - card handling equipment
   - printers
   - typewriters and teletypewriter terminals
   - displays and plotters
   - communications adapters
   - tapes
   - disk units
   - drum units
   - on-line sensors and output controllers
   - other computers
4. number of computer programs available
   - to support the machine operations
   - to support programmers
   - to solve problems
   - to support terminal users
   - to support system analysts

A characteristic of a commercial product line is the way in which the above facilities are coordinated so that a large number of possible configurations of the system can be used efficiently. In the typical special processor whatever support is provided in the way of balanced performance or software support is optimized around the original problem statement and may be somewhat clumsy when used for a variation of the problem or for a different problem.

* Some Class III problems require so many identical computers that economies of scale can be realized.

† Which may contain multiple CPUs.

The areas in which commercial and special machines can be roughly the same include:

1. Ability to multiprogram (time-share more than one program at a time);
2. Ability to multiprocess (run one problem on more than one computer to share the load);
3. Ability to handle real-time inputs;
4. Ability to connect redundant system elements through cross-channel switches;
5. Ability to interface with special processors of a different type;
6. Ability to nterface with special input/output devices.

The fact that commercial market requirements have grown to include these features makes the standard products highly competitive with special machines. However, there are performance limitations in many commercial machines which can be avoided in special machines at the expense of balanced machine operation. The areas in which special purpose processors can be superior include:

1. Instantaneous speed of input/output (bits per second per channel);
2. Maximum capacity of I/O channels (bits per second for all channels taken together);
3. Interference between memory cycles and I/O cycles;
4. Ability to do parallel processing (run many copies of the same program at once);
5. Do self-diagnosis and error correction.

With few exceptions, the special purpose processors used in military and federal civilian applications were built by the same people who build the commercial products. Therefore, there is no technical reason why the same capabilities could not be offered by them in their standard product line. Their decision not to make such products available is an economic decision.

Software and compatibility

As mentioned earlier, the computer programs required to construct and operate a complex system are beginning to dominate the project. Not only are the costs, as a proportion of project cost, increasing but the schedule impact of software development is being felt in later operational dates. The larger the programming task is, the more severe the impact is. In order to shorten the overall development schedule two steps are recommended:

1. Use existing tested software where applicable.
2. Build and test programs in parallel with other elements of the project.

These steps are facilitated by the use of commercial machines because the vendor can supply a library of tested support programs. Commercial test machines compatible with the proposed operational computer can also be obtained. The library of software for commercial machines extends beyond basic operating systems, languages, and utilities. Information handling systems which assist the user build and maintain files as well as query them are offered in versions which stress speed of retrieval and versions which provide great file handling flexibility at slower speeds. Management control support programs can be obtained which process project activity network data. Mathematical and statistical routines, output display formatters and report generators, flowchart programs, and various aids to the programmer are also found in vendor libraries.*

The process of programming consists of designing the logical set of steps to solve a problem and then coding and testing the steps. Both logical design and coding are subject to error; the errors, or "bugs" are detected by running program segments with test data. As the programmer finds and corrects the bugs he then repeats the process of design, code and test until his unit of programming is correct. Later his unit will be integrated with the programs written by others and the group of programs will be tested against the system specification. Without exception, additional errors will be detected at this point. However, it is not clear whether the new error is due to a faulty program unit, a faulty test case, a machine error, an operator's mistake, or a faulty system spec. Therefore, the analysis of the error involves a great deal of trial and error and a lot of machine time. It is very important that this process be carried out on a machine which acts exactly the way the final operational system will act if all the errors are to be found. Thus, the machine used by the programmers for debugging their program units should ideally be compatible with the machine in the operational system. Programmers need large blocks of uninterrupted computer time. Rather than sharing an engineering prototype, programmers want their own machine that will let them get started on productive work at the earliest possible time.

* Modern libraries include programs from many sources; some are available at no charge, others are sold, leased, or licensed.

Carrying this concept one step further, many programming managers are examining methods for increasing programmer productivity by giving the programmer better access to a computer and by giving him better tools at the computer. The access problem is addressed by installing a time-sharing system which allows the programmer to do his work at a terminal, cutting out the middlemen in the computer facility.* The tools question is addressed by providing programs to maintain the programmer's library and history files. With these tools, the programmer can create test data files, program files, and command files so that he can invoke a set of commands to execute a set of programs against specified test data. He can do this without handling any of the input data more than once, saving himself a lot of paperwork and a lot of running back and forth to the machine room. When the machine he is using is compatible with the final operational machine, it is a simple matter to transfer the set of completed work to the operational system to signify that he is done. When the final machine is different, there is an extensive requirement for translation, reprogramming, and retesting before the program can be accepted for the final system. In other words, *the compatibility of the support machine with the final machine shortens the time and lowers the cost of producing a final version of the programming system.*

The benefits of the compatibility concept carry over to the operational system in some instances. Several government systems have been specified in such a way as to have all programming done at a central point. In TACFIRE, an Army artillery fire control system, the programming was to be done in the U.S. and shipped to tactical installations at field locations around the world. The programs, written in PL/1, would be debugged and tested on a commercial machine in the U.S. using a compiler modified so as to create output programs in the language of the field machines. Under these conditions, it must be assumed that no changes of any type were to be made to the programs in the field. Such a system is feasible but is more difficult to manage than one in which the ability to make a change in the field exists. Compatibility between the base machine and the field machines would provide that option in systems similar to TACFIRE.

A considerable advantage of commercial machines for the fast startup of a project and for the installation of the initial capability lies in the ease of training the project staff to use the equipment. The likelihood that trained personnel can be hired to use commercial machines is high and getting higher as schools add data processing courses to their curricula. The likelihood that trained people exist for a special purpose machine is nil. Furthermore, the nature of the training for a commercial machine is identical to that offered by the vendor or any number of independent computer training schools. The training program for a special purpose processor is almost always unique and may be unsuitable for teaching institutions. The corollary of the training problem is that many people are willing to work on a general purpose machine whereas they are reluctant to accept employment to work on a unique machine. Their reasons are tied to their future career plans. Employment history shows that a computer engineer or programmer has more opportunities for advancement when he is in the main stream of the industry that when he is associated with a special purpose one-of-a-kind system.

Hardware and compatibility

There are several other factors affecting the specifications of military and federal civilian systems. Some are spurious; for instance, it is not unknown for a buyer to specify an advanced componentry because it is new—not because it is essential. This was common during the transition from vacuum tubes to transistors. Although such a spec may have been necessary to achieve a performance objective, in some situations it simply increased the risk of on-time delivery. Other specs are more significant because they reduce the system cost and reduce the complexity of the system both functionally and operationally. An example is the specification of a standard interface for connecting input/output devices to the computer. At the many Air Force and NASA ranges, for instance, there was a proliferation of special sensors to track rockets and aircraft and to process communications from space vehicles and from remote tracking stations. Acquired at different times for different purposes, the sensors had a wide variety of interfaces. Consequently, when it became evident that unified range operations would improve the support provided to all range users, a whole new set of "black boxes" had to be acquired to connect the sensors into the common network. In this respect, commercial machines are ahead of the government. Standard interfaces that allow off-the-shelf equipment of one manufacturer to interface with that of another manufacturer are the norm. Vendor specifications of their standard interface in terms of wiring diagrams and voltage levels are readily available so

* For simple debugging, the time-sharing system may look like a subset of the operational machine. The use of "virtual machines" can eliminate this restriction if required.

that it is reasonably easy to procure peripheral devices that meet the standard. This facilitates the use of other vendors' devices and permits the attachment of special purpose sensor and control devices to the central computer. The result is a capacity for horizontal growth.

Manufacturers provide for vertical growth by designing a compatible family which includes central computers of various speeds. Thus, the IBM S/360 line provides for compatible processors (each capable of reading the same programs and handling the same input/output records) that can handle anywhere from 5.33 bits per microsecond to 2370.37 bits per microsecond. While the bits per microsecond do not relate directly to performance, they do give a feel for the very wide range of capability found in a compatible family. Additional vertical growth can be obtained by increasing the amount of storage available on the central computer. This contributes to throughput by reducing the amount of traffic between the central computer and the auxiliary storage devices. Here, too, the range is enormous running from a few thousand characters to several million characters of storage in the same family. An analysis of the combined computational and information handling capability of a line such as the S/360 shows a ratio of overall performance of 36×10^6 between the very large S/360-195 and the very small S/360-20. More realistically, there is a range of two orders of magnitude in the performance of the Model 195 versus the Model 65 which would be a typical development support machine for the large processor.

The existence of compatible families of commercial machines permits the installation of a small machine for which complete software support will be available in more or less the same form as the support for a larger version of the machine so that the scaffolding developed around the small machine is directly transferrable to its successor. Thus, maximum use can be made of commercial components early in the schedule without compromising the flexibility of the system design.

Reliability and serviceability

Under the topic of reliability and serviceability are included considerations of availability (is the machine ready when requested?), reliability (does the machine function correctly?), maintainability (how long does it take to service and repair?), and modifiability (how easy is it to add a feature to correct a recurrent problem?). In engineering terms, these characteristics are often provided by redundant facilities. In some machines, the redundancy is built into each box by repeating circuits twice or three times. More commonly, the system as a whole is redundant with more than one processor connected through cross-channel switches to multiple copies of the input/output devices. When one element fails or deteriorates, it is removed from the system for maintenance and the rest of the system continues to operate. The level of redundancy is established so that the remaining equipment can handle the entire application (fail-safe) or some subset of the application (fail-soft). In addition, the software is designed to permit frequent checkpoints which copy the status of the machine so that, if a subsequent error occurs, the program can back up to the checkpoint and restart with correct data. The time required for automatic restart is small compared to the repair time of the machine and, in fact, the operator may not even notice that an interruption occurred. The ability to do these things was developed on special purpose machines but is now generally available on all major commercial machines.

The reliability which is demanded of individual units of equipment may exceed the design limits of commercial hardware. This degree of reliability, however, may be somewhat unrealistic. Enemy action, failure of sensors, or natural catastrophe might cause a system failure quite independent of the condition of the individual system elements. For instance, suppose an air traffic control system were based on data from satellite-borne sensors passed to the ground through a communications ground station. In principle, one ground station would suffice for the system. In practice, one ground station is inadequate because it could be put out of action by an earthquake, fire, aircraft accident, broken transmission line, etc. At least two ground stations are needed to make the system reliable. The additional ground stations could share the system workload or could act as backup for the prime station. In each station there could be a duplex or multiprocessor computer to increase its availability. Now the system as a whole is more likely to work properly than if there were only one station regardless of the reliability of the individual units in the single ground station. Thus, although commercial machines fail to meet the initial hardware specs of many special applications, the need for these restrictive reliability standards can be called into question in those applications where system reliability results from good system design.

Component technology in commercial hardware equals or exceeds the performance called for in contemporary special hardware. The large volume of the commercial vendors had led to the development of mass production techniques for componentry which have a high yield at very low cost. The comparable process technology for special components would require the same initial development expense without the equivalent product base against which to amortize the cost. A related benefit of mass production is that the lifetime in the field for a given family of components is quite high. This makes it economically feasible for a manufacturer to continue to produce spare parts for that component family long after new production of the machine is terminated.

The fact that commercial hardware does use production quantity components leads to a very high degree of standardization with the system. The number of different parts used is generally held down to facilitate the logistics of resupplying the field maintenance staff. A great deal of effort goes into making the system simple to maintain: extensive diagnostic aids are provided, pluggable replacement assemblies of optimum cost (trading off repair versus throwaway) are designed, multiple sourcing for parts is planned, training material is designed to be explicit yet easy to learn and understand, machines are assembled for easy access to components, etc. Commercial machines are built, wherever possible, to use standard power sources and to place a minimum burden on the user for special office conditions or construction. The commercial system development cycle includes the preparation of training programs for maintenance people as well as engineers and programmers. In most cases, a pool of experienced maintenance people exists at the time of delivery of the first field machine. Similarly, a large and adequate stock of spare parts and replacement units is placed on the shelf to meet the needs of the users. The byproducts of mass production are not obtained in special purpose computers. Here, reliability is obtained by more careful parts inspection, by designing for larger margins of safety, and by continuing operation while minor repairs are in progress. The degree of reliability achieved approaches that desired as close as the buyer's budget permits. The net result is that, in areas of service and reliability, the commercial standard is very high and, although not higher than what the government buyer wants, it is often a good deal higher than what he ends up buying.

Flexibility

The use of standards to improve the general purpose product could restrict the flexibility of commercial systems. Fortunately, this has not occurred. The standards normally apply to components and interfaces in such a way that system modification is quite easy. This can be observed in applications where one aspect of the problem must be processed at a much faster rate than other aspects. As an example, a communications satellite may store data received as it travels around the globe and then spew it out all at once when it passes over a receiving station near its home base. The data must be accepted when the satellite is overhead because there will never be another opportunity to pick up the message. In this case, commercial machines can still be applied as, indeed, they are. A special input buffer is required to collect and hold the satellite output; the buffer is a very high speed recorder. Then, at computer speed, the buffer transfers the data to the computer. In some instances, such as at missile ranges where the data stream is to be sampled before it is analyzed, the buffer may do a limited amount of data screening, editing, and summarizing before transferring the stream to the computer. This combination of an "intelligent channel" with a computer is a special case of the commercial machines' ability to interface with special I/O devices. When the input stream exceeds the computer capacity for extended periods of time, it is necessary to replace the computer with a higher speed processor. In practice, this requirement occurs with both commercial and special processors; however, the replacement is often more practical with commercial machines from compatible families.

In those problems that call for a parallel processor, several options are available. Very few problems are so well structured that, at all times, they can be subdivided into exactly the same number of identical computations. Thus, a machine like the Illiac which provides 64 parallel processors may be fully utilized only when a 64 part problem is being processed. Most problems involve some serial processes in addition to the parallel processes. For instance, in a system which analyzes seismic data to detect earthquake events, the process may start with some housekeeping to set up the input record format and allocate some memory locations to the results of the analysis. Then, by applying a transform algorithm to the input data the results can be computed and stored in the allocated space. The frequency with which the computer is asked to compute the transform may be small in the light of other jobs to be done in the facility. Therefore, instead of buying a parallel computer, the project manager will consider augmenting a standard commercial machine with a low cost Fast Fourier Analyzer box that attaches to an input channel of his computer. The speed of the FFA is only used for FFA problems; the rest of the time it is ignored.

LIMITATION OF COMMERCIAL HARDWARE

The advantages listed above are not without limits. Several limiting features of commercial machines have been mentioned but bear repeating. Others related to commercial business practice are also relevant.

Environment

A typical office environment implies a room with air conditioning and good janitorial services, plenty of enclosed storage space, space for maintenance people, etc. Such facilities are often available for military systems even when the room is inside a bombproof shelter. When the computer is in a properly shielded environment, a special effort to "harden" the computer against shock, vibration, fungus, vermin, dirt, electromagnetic and nuclear radiation, etc., is redundant. On the other hand, some machines are meant to be used on board ships, in jeeps, in aircraft, in remote locations without normal stateside services, etc. In these cases, the hardware itself may be designed to be protected against the hostile environment. The hardening takes the form of specially built components which, in general, cost more than those used in commercial products; therefore they are not found in off-the-shelf equipment except for the machines aimed at the process control market (steel mills, paper mills, refineries, etc).

It has been demonstrated many times that commercial hardware can be installed in military vehicles so that it can be transported without damage and can be operated in the vehicle given an appropriate enclosure. Transportability requirements are not uniform, however, and the degree of transportability varies from machine to machine. The most powerful machines tend to exceed the floor area of one van. Certain general trends have improved the ability of the industry to build transportable machines: smaller boxes that are easier to place in a vehicle, more liberal temperature ranges, etc. These features are evident in the reduction of setup time for a commercial machine. In the 1950s a week was allowed to assemble a large computer and check out the problems that may have occurred in the move via padded van from the factory. In 1970, a day or two is all that is required with very few problems introduced by the transport. When the same machine is to be placed in a van—as has been done in the Army CS-3 system—some additional precautions are required. Most of them are mechanical. Stiffening of the cabinets and the movable panels in the machines may be required to reduce the effects of shock and vibration. The whole machine may be mounted on shock absorbers to shield it from the environment. And a short checkout cycle may be added to the set-up procedure after the van comes to a stop and the machine is turned on. In general, the commercial machine is not operated while the transport is moving (except on board ship).

The input/output devices for the commercial market present a more severe problem in transportable systems. They are primarily mechanical devices with sensitive alignment and close tolerances between high speed moving parts. They do not stand up as well as the electronic computer. It is normal for the project manager to substitute militarized I/O devices for the commercial products in transportable installations. Specially designed disk and tapes have been built for this purpose and, by virtue of the standard interface concept, can be attached to a variety of computers. A transportable system can consist of all off-the-shelf hardware or a combination of standard processors and special peripheral devices. Thus, it is not essential to assume that all elements of a military system must be hardened. If a commercial machine is the best one for the application, it can often be installed in such a way as to be protected from the environment.

When it comes to the third class of applications where the environment is so hostile that special hardening is required, the commercial machines are no longer feasible. Machines in Class III are usually very small both physically and functionally. They are built to fit into odd-shaped spaces. They weigh very little. Their word sizes and instruction sets are smaller than those offered in commercial machines. Their input and output devices may bear no resemblance at all to the cards, tapes, and disks found in larger information handling systems. As an example, the computer in a space vehicle may be used only to calculate orbital rendezvous parameters or descent speeds and angles. Its inputs may be from instruments in the cockpit of the capsule; instruments used nowhere else except in this particular space mission. Its size may be that of a home tape recorder to permit it to fit in the astronaut's console. Its memory may be only large enough for a single computation; a new computation may require reloading of the memory by radio from an earth station. The computations it performs may be so well worked out in advance that the magnitudes of all the numbers in the computations are known; therefore, there is no need for floating point instructions in the repertoire of the computer. The programs are probably written and optimized in machine language so as to take as little memory as possible; therefore, no high level language compiler or massive operating system is called for. All these features tend to make the general purpose capabilities of commercial hardware and software unnecessary for the space computer. In fact, the bulk represented by the standard features of commercial hardware is undesirable in the space computer. On the other hand, it is essential that the space computer work when it is needed. This leads to a high degree of redundancy in the design of the system in order to increase the reliability, availability, and fail-safe characteristics of the system. Since, in the example of the space capsule, there is no practical way to maintain or repair the computer, it is possible to design the computer to be discarded on failure and replaced by a spare. Obviously, the priorities for a system of this type are quite different from those set for commercial computers. These *priorities make specialization an asset in the third class of application where it is a penalty in the other two classes.*

On the other hand, some applications in Class II call for the same environment as Class III even though they require the functional capabilities of Class II machines and programs. Examples are surface and undersea naval applications, airborne applications or airborne back-up for ground based systems, and long duration manned space laboratory missions. Some of the applicable specifications might be:

1. Dimensions — 26″ × 48″ × 66″ for passage through watertight hatches
2. Weight—45# per box for 2-man lift
3. Temperature and Humidity—operate in shelters with interior temperature 60°-90°F, relative humidity 15-95 percent, operate for limited periods at −65° - +125°F, non-operating exposure to −80° - +160°F.
4. Other—withstand salt, fog, rain, sand and dust, sunshine, fungus, high altitude, 5g vibration, 30g shock, vermin.

It is not cost effective to design normal commercial hardware to withstand these extreme conditions. However, it is possible to build hardened machines to specs that are compatible with commercial machines. This will permit the benefits of commercial support in the development process to be realized. It may also facilitate communication between hardened systems and normal systems; i.e., a ground based command center and its airborne alternate.

Commercial practice

The project manager who is convinced that commercial data processing is appropriate for his project may find that there are some obstacles to his getting what he needs. The practice in the industry is to withhold information on advanced developments until the machines are fully tested and are ready for public announcement. Prior to this time, the existence of a suitable machine may be unknown to the project manager. His best move in these circumstances is to make his needs known through an industry briefing. Then, by agreeing to protect the proprietary rights of the vendors, he can obtain the advanced information for use in his decision process.

There is a tendency for government project managers to schedule the system so that they get to install the very first machine of a new product line. Although the vendor does a very thorough job of factory test, the first one or two machines off the production line are still liable to have bugs in them. The project manager would be better off to downgrade the glamour of the first installation and place more weight on the stability of the later machines in the production line.

A commercial machine is suitable for a government system when it meets all the requirements of the government system. However, the commercial machine was designed for a broader market than the single government application. It contains many features in addition to those needed by the project. There are two disadvantages and one advantage to this. The cost of the unnecessary features is included in the price of the machine. The cost is not restricted to the acquisition price of the machine. The added features also take up space and use up time in the operational system. This disadvantage is offset by the fact that the commercial machine is usually much less expensive than the special purpose alternative. In some critically timed real-time systems, the added features must be removed. This problem ties in to the second disadvantage. The more features a machine has, the more complex it is and the more things that can go wrong with it. The effects of complexity are controlled by good system development procedures but their impact on operations and maintenance cannot be ignored. The advantage of the added features found on commercial machines is that the programmers can usually find ways to use the features to improve the system software.

There are other features that are generally useful in government systems which are seldom found in commercial machines. The average business data processing application does not call for them. An example is the ability to remove a unit of the computer for service without interrupting the computer operation. The ability to reconfigure around a failed unit has been provided with some commercial systems but they are designed to stop while the unit is removed and then restart with some manual intervention to describe the new configuration. In systems with enough redundant elements to permit continuous operation there is no reason to stop the computer to remove a unit; however, special modifications may be needed to prevent the computer from automatically stopping under these conditions. The modifications essentially place a switch in the hardware that isolates the failed unit electronically the same way a switch in the software permits the logical rearrangement of boxes.

When the project manager recognizes the need for a modification to the commercial machine there is generally no great problem in designing and installing the change. However, the industry business practice may increase the cost impact of that change. The normal maintenance of commercial hardware and software has been designed to give maximum service at minimum cost. This is achieved by standardizing the training and diagnostic aids given the service personnel. A corollary of this approach is that the serviceman is not well equipped to understand user modifications to the hardware or software; therefore, his service commitment may be modified or even voided by certain types of modifications. The protection against loss of service is simple; the project manager and the service supplier should mutually agree to a service agreement which spells out the extent of the service commitment and alerts the manager in advance of the exposure he has. The reverse of this situation may have similar impact. The vendor of commercial hardware normally makes engineering changes to the machines over a period of years. The purpose of the changes is to correct minor problems, facilitate service, reduce failure rates, etc. The changes are transparent to the user except in cases where the user has modified the hardware. The project manager must be prepared to trade off the benefits attributed to the engineering change against the value of his modification.

CHARACTERISTICS OF THE DEVELOPMENT PROCESS

The systems included in the second class above are dominated by large real-time information handling systems—SAGE, Air Traffic Control, SAFEGUARD, TACFIRE, Naval Tactical Data System, Marine Tactical Data System, Apollo, Orbiting Astronomical Observatory, National Military Command System, Intelligence Data Handling System, etc. *The consequence of size is complexity.* There are so many interactions among the elements of a large system that no one man has the ability to comprehend and manage the total entity. As a consequence, the system is organized into subsystems and smaller components to limit the span of control of each manager to a scope that he can handle. The step by step delineation of system components changes the focus of attention from the system itself to the parts of the system. As a result, the structural interactions which define the system may not get enough attention to ensure their integrity.

Project control

The concept of "change" is basic to system development. It is unrealistic for a system manager to assume that the objectives established at the outset of a project will be unaffected by changes in the environment or in the detailed plans for implementing a given function. Rather than be crippled by an inflexible plan, the manager establishes procedures which permit the formal modification of his baseline specification by changes which are agreed to by all affected parties. Such a set of procedures protects the system against arbitrary changes but permits necessary changes. In order to be effective, the change control process must be able to determine the interactions of each proposed change with the rest of the system.

The government calls the control procedure "configuration management" and provides vendors with guidelines for describing the work breakdown and keeping track of the base specification. For a large system, it takes about ten years to reach the operational phase of the cycle. Lots of things change during that period which can upset earlier commitments. Brigadier General W. R. Reed, U.S. Army, suggested how serious the problem can be when, after eight years of a study and design effort, it was realized that changes had occurred in computer technology that had a profound impact on the environment being studied. In fact, he estimated that computers improved by a factor of ten every 3.8* years. Such an order of magnitude improvement required a new definition of the environment in which the computers were to be used. Gen. Reed was talking about the Army tactical fire control system, TACFIRE, which in eight years had been obsoleted twice before it ever reached the acquisition stage until 1968 by which time its specs were again being overtaken by new technology. The guideline implied by this situation is that computer specs should not be frozen early in the system definition stage since they will probably have to be redone later.

Postponing the actual computer decision does not impede the progress of the system. The data processing subsystem can be described in terms of the functional and performance characteristics desired. These characteristics can then be incorporated into a model of the subsystem which can be exercised to evaluate the design concept and predict the performance of the actual hardware when it is selected according to the

* As compared to my estimate of 5 years. The rate of computer improvements is hard to determine but remarks by Lowell D. Amdahl, president of COMPATA, Inc., at the June 1970 meeting of the IEEE computer group indicated that Gen. Reed's figures have not changed substantially since 1962.

approved design. The model is a computer program which can be run on a standard commercial machine. (This technique, used with great success in the Apollo program, has added value when the computer used to exercise the model is compatible with the computer ultimately procured for the system. When this is the case, parts of the model may be reusable in the ultimate system.)

The model is a design aid and may be thrown away after the total system goes into operation. Preferably it is retained as the basis for continuing improvement of the operational systems. Nevertheless, it is never more than a part of the scaffolding required to support the system builders. Other parts of the scaffolding for a data processing subsystem are test programs and data generators; input/output simulators; computer services for running engineering problems and debugging programs; compilers, sorts, and other programs to prepare system programs for execution; controlled libraries to separate work in process from released work; project control administration and analysis (as in PERT); and many other activities both technical and managerial. *The optimum environment—from the standpoint of project control, training, cost, and testing—is one in which the same computer family is used for all functions of control, support, and ultimate operations.* The advantage of sticking with one computer family throughout the life of the project is that predictable results can be achieved on schedule without the distractions posed by a new, incompatible machine.

System development guidelines

Various guidelines are generally accepted as constructive when dealing with large systems. One such guideline attempts to strengthen the system structure without tying the hands of the component developers. It calls for the design to be done from the top down. That means that the structure of each level of the system is designed before the details of the next lower level are developed. Top down design is an effective device for identifying and controlling the interfaces within the system. In addition, this approach minimizes the constraints on the component designers because it provides them with a set of functional and performance objectives for their area of responsibility without telling them how to build their specific "black box". Since each element of the system is simply a "black box" to the level above it, a different black box with the same external characteristics can be substituted for the original element without making any other changes to the system. This is one reason why it is feasible to use a simulated environment during the development of a system; simulated black boxes are used until the real thing is available. Such design flexibility is essential during the design stages of the system specifications.

Phased implementation

Aware of the need for adapting to changes in the system design, the system requirements, and the system environment, experienced system managers lay out a phasing plan which will reach the final system in useful steps. They start with an initial operating capability (IOC) to get a minimal capability for meeting operational requirements and to provide an environment in which to test out the design concepts for the operational system. Only by installing an IOC can the designers evaluate their ideas against realistic man-machine tests and modify the final system to overcome the weaknesses observed in the IOC. The existence of the IOC often leads to more pervasive changes to the system design as well. Considering the long lead time for most large systems, the original requirements statement is often out of date when the IOC is installed. Top management does not realize this until they try to use the IOC. Then they observe that some of the things they asked for are not appropriate to the problems they face today. In addition, for the first time they have an opportunity to observe how the data processing system can contribute to their decision making functions. This leads them over a period of time to restate the functions assigned to the computer so as to take advantage of its speed and flexibility. Invariably, their new definitions of the computer workload place more work on the machine and less on the user. Therefore, it is normal for the final operational system to represent an upward growth from the IOC.

Recognizing that the IOC is a test vehicle, not the ultimate system, it should be installed at the least expense but in a form that is easily modified. The best way to minimize the cost of the IOC is to acquire a commercial computer. In this situation, the original design and development costs have been borne by the vendor and are distributed equitably among all the users of that computer. A special purpose machine in the IOC passes all the design and development costs as well as the manufacturing cost along to the single buyer.

The decision to select off-the-shelf hardware for a government system affects the balance of the system plan. The timing of the decision to use commercial data processing equipment should be made in the system definition phase of the development cycle. It should occur after the overall system design and all of the subsystem functions have been defined and before the subsystem component specifications are complete. Having decided to use commercial technology, the actual selection of a machine should be deferred until the start of the acquisition phase to take advantage of the latest available technology. As stated earlier, the system specifications can be written around function and performance objectives rather that detailed characteristics of the data processor so that the delayed procurement does not slow up the design process. By selecting a machine that is part of a compatible family of computers, the manager can allow for future growth requirements. To achieve the benefits stated earlier, the manager must verify that the compatible family includes basic software packages, that the maximum capacity machine in the family is large enough to permit substantial vertical growth, and that the family has a published standard interface.

SUMMARY AND CONCLUSION

Most of the performance and reliability specifications for military and federal civilian applications can be handled by one or more machines available off-the-shelf. With minor modifications to their functional capability, addition of special I/O boxes, and installation in suitable enclosures, the commercial machines can generally fit all the requirements of ground-based application areas. In fact, because of the family characteristics of many commercial machines, they outperform the typical special purpose machine with respect to growth capability and cost effectiveness. The delivery schedules for commercial machines are substantially more reliable than those for specially engineered equipment. Although the first machine delivery off the production line in commercial practice may vary from schedule by several months, subsequent deliveries are sufficiently certain to permit vendors to accept penalty clauses for late delivery as a normal business practice. By comparison, special machines have been as much as three years late with delays of one year from the projected schedule occurring with some frequency. The reason for the delays is seldom poor workmanship on the part of the builder. Most of the time it is due to the fact that changes to the machine specs occur during the machine development cycle which cause rework and rescheduling. The commercial machines have been fully specified before they are announced and before delivery commitments are accepted; therefore, the specs are not subject to change in the same way. The price of commercial machines is more stable than the price of the typical cost-plus special machine; again, because of the different impact of product changes. More important to the integrity of the application, commercial machines have already passed stringent product tests before they are announced for sale. The project manager can have very high confidence that a machine that meets his requirements on paper will perform properly in fact. He cannot have the same level of confidence in a special purpose machine since he must commit himself to its design specs without any assurance that the machine so specified can be built and, if buildable, will perform exactly as planned. His ability to predict these things at procurement time simply falls far short of his ability to predict—and measure—the same things on off-the-shelf systems. At the same time, he is able to compare his requirements to the offerings of several competitive vendors in the commercial market in order to select the least expensive system for his project.

Risk reduction is a key advantage in the eyes of the project manager. In addition, the advantages in the development process are significant. The fact that a test cell can be established for the engineers and a separate support machine can be set up for the software team contributes to shortening the system schedule. The testing itself becomes less risky. Because the computer and its vendor software have been extensively tested before shipment, the application programmers can simplify their test detective work by assuming that the commercial data processing subsystem are error-free; therefore, errors must lie in the application programs or test cases. The assumption will not always be valid but the order it imposes on the error analysis procedure will speed up the process. In particular, the assumption that the various programs in the system have been fully debugged directs the attention of the test team to the interactions among the components where it belongs and away from the internal workings of each program unit.

Since systems are built and used by people, a major effort in any system development involves the training of the development team and the operational and maintenance teams. The earlier the training can start the better as long as the material taught is truly representative of the final operational system. When using commercial machines, the knowledge that the machine and its software are well defined permits training in this area to begin immediately after the procurement decision. Effective training can be procured from a variety of sources without the need for structuring a permanent educational office in the project. Moreover, the computer time needed for the trainees can be obtained on the vendor's machine (which, by definition, will be compatible with the project machine) without interfering with the test activities on the project machines.

The project manager concerned with meeting a set of operational objectives at minimum cost on a firm schedule must minimize the risk associated with each element of his system. By specifying commercial data processing subsystems he can satisfy his functional and performance needs in the large class of command and control applications as he has traditionally done in administrative applications. He will accomplish this with less cost, equal or higher performance in both the development cycle and in the operational installations, simpler logistics, and with much less risk of slippage or overrun. A general guideline to follow, contrary to past practice, would be that *commercial data processing should be used in all cases except those in which there is a specific obstacle to prevent them from doing the job.*

Programming language efficiency in real-time software systems

by J. O. HENRIKSEN

University of Michigan
Ann Arbor, Michigan

and

R. E. MERWIN

SAFEGUARD System Office of U. S. Army
Arlington, Virginia

INTRODUCTION

Early in the development cycle of a real-time software system, the designer must select one or more programming languages to be used in writing the system. As shown in Figure 1, there is available today a continuum of programming languages ranging from "high-level" languages like PL/I to "low-level" traditional assembly languages.

At the present time, high-level languages are widely used in conventional scientific and commerical software production, while low-level languages are widely used in the development of real-time systems. The wide acceptance of high-level languages in conventional applications may be attributed to favorable cost-performance tradeoff. Given an application whose data and program structures can in a reasonable straightforward manner be represented in a high-level language, use of the high-level language will result in conservation of human resources at the expense of machine resources. Real-time systems represent a class of applications where machine performance becomes a critical resource. In this case, factors other than fiscal cost of the machine resource become dominant. As a consequence, the program and data structure of a real-time system must be hardware motivated and any programming language used to implement such systems *must* allow the programmer to fully exploit the characteristics of the hardware. Since the design of program and data structure is hardware-motivated, artificial software restrictions, in the form of language limitations, are unacceptable. Such considerations have given rise to a new type of programming language, called "systems implementation language." The systems implementation language in effect attempts to represent a *range* on the continuum of programming languages. It permits the programmer to readily access and exploit hardware features, but allows him to perform the bulk of his coding by utilizing a less cumbersome higher level language. There are several attempts to analyze programming languages described in the literature.[1,2,3] In the sections which follow, an investigation is described in which representative programs from a phased-array radar real-time software system are coded in selected programming languages. Performance measures are described, results tabulated, and conclusions are drawn on the basis of these results. This investigation was carried out under the sponsorship of the SAFEGUARD System Office of the U.S. Army by Brown University,[4] the System Development Corporation,[5] University of Michigan,[6] and the Westinghouse Corporation.[7]

MODUS OPERANDI

Three existing programs developed to control the phased array radar used in the SAFEGUARD Anti-Ballistic Missile System test program on Kwajalein Atoll were selected as typical examples of real-time software. A functional description including flow charts and data sets was prepared for each of these programs based upon the actual assembly language code for the equivalent function produced for the SAFEGUARD data processor. From the supplied functional descriptions, the study participants constructed programs in various languages and ran standard sets of test data

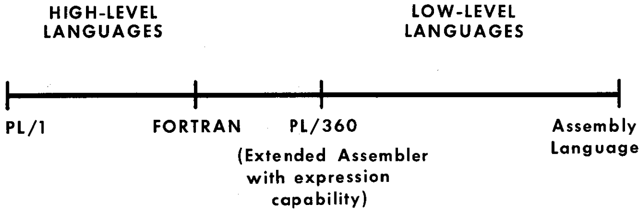


Figure 1—Continuum of programming languages

for performance measurements. Five languages; PL/1, FORTRAN-IV-H, JOVIAL, Brown University's Language for System Development (LSD), and IBM 360 basic assembly language were compared in performance runs on IBM 360-67 hardware. FORTRAN-V and SLEUTH assembly language were compared on the UNIVAC-1108. Of the languages compared, PL/I, FORTRAN-IV, and V are selected as representative high-level languages, JOVIAL as a precursor of systems implementation languages, LSD as systems implementation language currently being developed, and IBM 360 basic assembly language as a baseline for performance measurement.

ALGORITHM DESCRIPTIONS

GETTASK

The first algorithm modelled, GETTASK, is a task dispatcher for a single task list multi-processor. In the system studies, tasks are related in PERT-like networks such as the one shown in Figure 2.

In such a network, the completion of Task #1 allows the simultaneous execution (if 3 processors are available) of Tasks 2, 3, and 4. In general, a given task may have any number of predecessor tasks which must be completed prior to its running, and a task may have an arbitrary number of successors. In the case where more than one task is eligible to be run at the same time, a priority scheme is used to assign a task to a processor.

The implementation of such a precedence network is achieved by use of absolute enablement bits (AEBs) and conditional enablement bits (CEBs). Associated with each task is a CEB word, an AEB, and a successor list. Each element in a successor list contains a pointer to another task's CEB word and a mask to be logically ORed with the CEB word. In the example above, the successor list for Task #1 would contain pointers to Tasks #2, #3, and #4. Each CEB in a CEB word corresponds to a unique precedence condition. For example, the CEB word for Task #5 in the above network would have one bit to be set upon completion of Task #2 and another upon completion of Task #3. Each time the logical OR is performed, a check is made to see if all CEBs are set to 1, and if so, the AEB for the corresponding task is set to 1, indicating that the task is ready for dispatching. The system is general enough to allow enablement of tasks as a function of time or system conditions by allowing certain dedicated bits to be set only by the operating system.

EXLIGEN

The second algorithm modelled, EXLIGEN, is a portion of a radar control program execution list generator which converts requests for radar transmissions, according to time and type, into a fast access execution list format for later processing. Scheduling of transmissions is done on a cyclic basis: a request made in cycle n is handled by the scheduler in cycle n+1 and actually transmitted in cycle n+2. A master "menu" of allowable transmissions is available to several subsystems within the radar control program, and it is possible that these subsystems can make conflicting transmission requests. The list generation algorithm is carried out in two passes: in pass 1, a scan is made of all requests in order of increasing time, and requests are consolidated according to request type; and in pass 2, a list is constructed and ordered by priority of request type with certain requests of the same type consolidated into single list entries.

UPDATE

The third algorithm modelled, UPDATE, is a portion of a Kalman filter algorithm used to filter radar return data. On the basis of signal-to-noise ratio in the radar returns, weights are calculated to be applied to the current radar observations, and using these weights, a

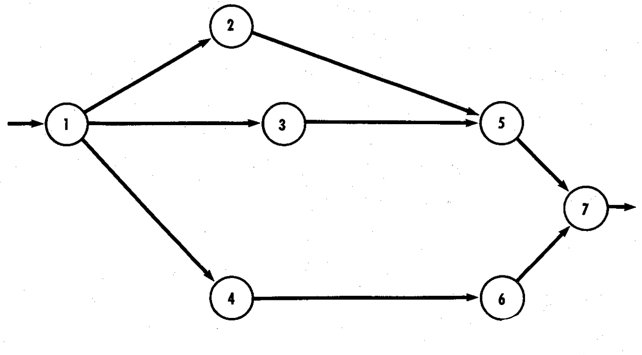


Figure 2—Task precedence lattice

state vector for an object in track is updated. The algorithm is completely described by recursively applied equations, which are amenable to solution in any FORTRAN-like language.

SUMMARY OF RESULTS

The results of the individual comparative studies submitted by the four participants are shown below. The four performance cirteria chosen were, number of source statements, number of object instructions, number of storage words required to contain the programs and data, and program running times. Comparative number of source statements reflect ease of expression in the two languages, i.e., a program that requires more source statements in one language than another is less naturally expressed in this language. Comparative numbers of object instructions are a partial measure of translator efficiency but do not reveal the impact of run-time routines called by the translator. Comparative object code storage requirements of language translators reflecting storage efficiency represents usage of a critical machine resource in most real-time applications. Specific comments on these criteria are included in the discussion of each study results.

The University of Michigan

The Michigan study was carried out on an IBM 360/MOD67-2 computer with 1.5M bytes of storage and compared FORTRAN, PL/I, and IBM 360 basic assembly language. In addition to coding the three algorithms with a "strict" interpretation of the original data structure, Michigan investigators carried out a redesign of the data structures used in GETTASK and EXLIGEN. The revised implementations, using redesigned data structures, were functionally equivalent to the original descriptions but exhibit enhanced translator performance.

Results, as presented in Table I, show a marked superiority of assembly language programs on the IBM 360. On the average for all cases tested, FORTRAN programs ran 2.7 times as long, and PL/I programs 7.4 times as long, as the corresponding assembly language programs. Results similar to this were noted in a benchmark study [8] which compared assembly language and FORTRAN on several large current systems. While the PL/I language offers a great deal of generality in the range of problems and data structures it can handle, it is at the expense of performance. Because of the wide range of the language,

TABLE I—University of Michigan Comparative Performance

	Data for FORTRAN IV-H and PL/1 Versus Assembly Language on IBM 360 MOD 67-2		
	Assembly Language	*FORTRAN-IV-H*	*PL/I* F-5
GETTASK (Strict Data Structure)			
Number of Source Statements	79	50	48
Number of Object Instructions (words)	75	251	327
Storage size (Bytes)	183	314	444
CPU Run Time (milliseconds)	0.70	2.84	6.74
GETTASK (Redesigned Data Structure)			
Number of Source Statements	67	39	48
Number of Object Instructions	65	170	292
Storage Size	83	244	413
CPU Run Time	0.49	1.78	3.14
EXLIGEN (Strict Data Structure)			
Number of Source Statements	259	122	133
Number of Object Instructions	228	477	1100
Storage Size	305	655	1205
CPU Run Time	1.83	4.98	19.20
EXLIGEN (Redesigned Data Structure)			
Number of Source Statements	228	126	130
Number of Object Instructions	200	405	783
Storage Size	276	557	1074
CPU Run Time	1.42	2.86	7.03
UPDATE (Strict Data Structure)			
Number of Source Statements	88	27	28
Number of Object Instructions	89	134	351
Storage Size	91	194	447
CPU Run Time	0.74	1.39	2.32

TABLE II—System Development Corporation Comparative Performance Data for JOVIAL vs Assembly Language on IBM 360 MOD 67 and Comparative Performance of JOVIAL on UNIVAC 1108 and CDC 3800 and IBM 360 MOD 67

		JOVIAL		
	Assembly Language IBM 360	*IBM 360*	*UNIVAC 1108*	*CDC 3800*
GETTASK (Strict Data Structure)				
Number of Source Statements	79	97*	77**	73**
Number of Object Instructions	75	198	119	158
Storage Size (32 bit words)	183	399	159***	154****
CPU Run Time (ms)	0.70	2.02	3.4	2.1
EXLIGEN (Strict Data Structure)				
Number of Source Statements	259	233	232	228
Number of Object Instructions	228	647	449	454
Storage Size	305	1146	566	566
CPU Run Time	1.83	4.47	2.0	2.72
UPDATE (Strict Data Structure)				
Number of Source Statements	88	41	41	41
Number of Object Instructions	89	133	96	102
Storage Size	91	171	40	40
CPU Run Time	0.74	1.79	0.78	1.52

* Includes 17 direct statements
** Includes 5 direct statements
*** 36 bit words
**** 48 bit words

producing an optimizing compiler is difficult, and the language has not had a long enough history to reach the state of development of compilers for other languages. One of the big disadvantages of FORTRAN and even PL/I is that the languages do not have syntactic forms which correspond directly to powerful hardware capabilities, i.e., the hardware is too highly masked by the use of these languages. For FORTRAN, the narrow range of syntactic forms appears to be the biggest weakness. On the whole, for the problems studied, FORTRAN appeared to produce reasonably efficient code for the statements allowed in the language, but the range of available statements was clearly restrictive to the programmer. The best performance of FORTRAN and PL/I was on the algebraic UPDATE program, which reflects that UPDATE was the most naturally expressed (in these languages) of the three problems.

The improvements in performance obtained by redesign of the data structures are interesting to note. For all languages, data structure redesign yielded performance enhancement, but the improvements obtained were substantially greater for FORTRAN and PL/I. This simply illustrates the fact that performance in high-level languages suffers as program and data organizations become more and more removed from basic structures in the languages. Since program and data organization in real-time systems are often dictated by hardware organization and data formats, applicability of high-level languages may be relatively low.

SDC

The SDC Study was performed principally on an IBM 360/MOD67 and additionally on the UNIVAC 1108 and CDC 3800 using the JOVIAL language. The goals of this study were to exhibit performance for another language on IBM 360 equipment and to explore the ease of transferability of software between dissimilar machines. The JOVIAL language exhibits a great deal of flexibility in definitional capabilities for data structures. It represents a substantial improvement over FORTRAN in this respect, but is not quite as wide ranging as PL/I. The JOVIAL language also allows a programmer to insert assembly language instructions into a program in order to utilize the hardware features he may desire. JOVIAL does not have (in

TABLE III—Brown University

Comparative Performance Data for LSD vs FORTRAN IV, PL/I and Assembly Language on IBM 360 MOD 67-1

				LSD	
	Assembly Language	*FORTRAN IV*	*PL/I* P-5	*I*	*II*
GETTASK (Strict Data Structure)					
Number of Source Statements	79	50	48	50	
Number of Instructions	75	251	327	159	
Storage Size (Bytes)	183	311	444	199	
CPU Run Time (ms)	0.70	2.84	6.74	2.2	
EXLIGEN (Strict Data Structure)					
Number of Source Statements	259	122	133	126	
Number of Instructions	228	477	1100	440	
Storage Size	305	655	1205	677	
CPU Run Time	1.83	4.98	19.2		
UPDATE					
Number of Source Statements	88	27	28	31	29
Number of Instructions	89	134	351	159	151
Storage Size	91	194	447	133	120
CPU Run Time	0.74	1.39	2.32	0.78	0.78

the IBM 360 version) optimization features which are as extensive as those of FORTRAN-IV-H, but produces "reasonable" code, certainly much better than that of PL/I. Performance of IBM 360 JOVIAL is given in Table II. The numbers shown may be meaningfully compared with the University of Michigan numbers for the strict data structures. As one might expect, JOVIAL performed better than FORTRAN-IV-H, much better than PL/I, and worse than IBM 360 basic assembly language in the GETTASK and EXLIGEN problems. In the UPDATE problem, however, the optimizing features of FORTRAN-IV-H enabled it to outperform JOVIAL.

The algorithms were executed on the UNIVAC 1108 and CDC 3800 by simply changing the data definition statements to account for different word sizes. Additionally, driver programs had to be altered because of differences in system input-output and timer routines. The significant fact is that with the exception of "direct" assembly language statements in GETTASK, no executable statements in the programs required change. The framework for carrying out such a transfer to a different machine is relatively easily used, as illustrated by the fact that the CDC 3800 conversion was done as a minimal effort by a programmer unfamiliar with the original IBM 360 JOVIAL version.

Brown University

The LSD language is being developed by Brown University as a systems programming language for the IBM 360. LSD is syntactically similar to PL/I, but not nearly as broad in scope. It is specifically tied to the organization of the IBM 360 hardware, for example, the 16 general purpose registers of the IBM 360 can be directly referenced and manipulated in LSD statements. The design philosophy of LSD can be summarized by saying that it attempts to make the hardware readily accessible to the programmer while retaining the ease of usage of a high-level language. In order to produce optimal code, the programmer *must* be thoroughly familiar with IBM 360 hardware, as only the simplest forms of optimization are performed by the compiler.

The Brown University investigators used University of Michigan implementations of the three programs as a starting point. PL/I statements were then transliterated into LSD statements by means of an automatic transliteration program. Results for compilation of the resultant LSD source code are shown in Table III under the column labeled LSD-I. The data generated for EXLIGEN by LSD could not be verified and should be regarded as tentatine. Next, the investigators carried out an optimization of the LSD code at the source

TABLE IV—Westinghouse

Comparative Performance Data for FORTRAN V and SLEUTH (Assembly Language) on UNIVAC 1108

	SLEUTH	*FORTRAN V*
GETTASK		
Number of Source Statements	212	50
Number of Object Instructions	173	192
Storage Size (in 36 bit words)	205	256
CPU Run Time (milliseconds)	1.2 ms	1.2 ms
EXLIGEN		
Number of Source Statements	673	177
Number of Object Instructions	645	746
Storage Size	684	828
CPU Run Time	2.6 ms	2.6 ms
UPDATE		
Number of Source Statements	228	35
Number of Object Instructions	212	181
Storage Size	227	212
CPU Run Time	.53 ms	.35 ms

level for the UPDATE program, taking advantages of facilities not provided by PL/I. The results are shown in column labelled LSD-II. University of Michigan FORTRAN-IV and PL/I results are included for purpose of comparison. Timings shown for Brown University results have been adjusted upward by 10 percent from the measured values on an IBM 360/MOD67I-1 in order to make them directly comparable to the timings obtained on the University of Michigan IBM 360/MOD67-2 (1.5M Bytes Storage).

Westinghouse

The Westinghouse study was performed on the UNIVAC 1108 and compared with the SLEUTH assembly language with FORTRAN V, UNIVAC's highly-touted optimizing FORTRAN compiler. Two programmers were employed, one coding only in FORTRAN and the other only in SLEUTH. They designed, together with the project supervisor, compatible data structures so that a single driver program could drive either the FORTRAN V or SLEUTH versions of a particular algorithm. The data structures were amenable to problem solution on the UNIVAC 1108 and were not the same as data structures originally developed for the SAFEGUARD data processor.

On the whole, the performance of FORTRAN V as shown in Table IV was of surprisingly high quality. The design of compatible data structures for SLEUTH and FORTRAN V appears to have been slightly slanted toward FORTRAN, i.e., to force the SLEUTH programmer to use a data structure that can also be used by FORTRAN (especially in the areas of manipulating data pointers) is somewhat of a handicap. For the UPDATE algorithm, which was purely algebraic in nature, relatively good performance was expected from FORTRAN. The algorithm was a relatively small piece of code and was simple enough to be easily handled by a compiler, i.e., assembly code "tricks" could not be employed to any significant advantage. In actual performance, the FORTRAN version ran significantly better than the SLEUTH version. If the SLEUTH version were to be recoded using the FORTRAN version as a guide, it is probable that it could be made to perform better than the FORTRAN version. The significant fact is that in the time frame allotted to this study, FORTRAN actually produced a better program, reflecting that in certain cases a cleverly written compiler can actually surpass human performance.

CONCLUSIONS

The following conclusions relative to the performance in terms of efficient object code production and CPU running times for programming languages, i.e., high-level or assembly, are drawn from the results summarized above.

(a) If CPU running time and storage utilization are the primary criteria for evaluation, assembly language coding is distinctly superior to high-level languages available today.

(b) System's Implementation Languages, which appear to offer the advantages of high-level languages with respect to reducing software development costs, also produce object code with efficiency approaching that of assembly languages. This performance improvement, largely achieved by allowing programmers to take advantage of system hardware features, implies a machine dependence of such languages and loss of transferability of programs written in these languages from one computer architecture to another.

(c) Careful design of program data structures can greatly improve the performance of present-day higher-level languages in production of efficient object code.

(d) The production costs associated with high-level languages are lower, up to a threshold at which specific desired manipulations cannot easily be made in the language.

(e) Transferability is easily achieved in high-level languages, provided that direct usage of machine dependent characteristics is carefully controlled.

(f) Data structures designed for a particular machine architecture must be modified if any kind of efficiency is desired in transferral to another machine architecture.

(g) The "ideal" high-level language must allow flexibility in data definition capabilities as JOVIAL does and FORTRAN IV does not, but be realistic in scope, as PL/I is not.

ACKNOWLEDGMENTS

The programming language efficiency study was carried out by four groups under the overall guidance of the authors. The study directors were Prof. B. Arden, University of Michigan; Mr. P. Waszkiewicz, Westinghouse; Mr. R. Knight, SDC and Prof. A. Van Dam, Brown University. The coding was done by Mr. J. Hamilton, University of Michigan; Messrs. G. H. Rouillard and R. L. Specht, Westinghouse; Mr. R. Brunstrum, SDC and Messrs. R. D. Bergeron, J. D. Gannon and J. V. Guttag, Brown University.

REFERENCES

1 F J CORBATO
PL/1 as a tool for system programming
Datamation May 1969

2 H HESS C MARTIN
A tactical command and control subset of PL/1
Datamation April 1970

3 R RUBEY
A comparative evaluation of PL/1
Datamation December 1968

4 A VANDAM R D BERGERON J D GANNON J V GUTTAG
Programming language comparison study
Brown University Report

5 R L BRUNSTRUM
A study of JOVIAL programming language effectiveness for SAFEGUARD programs
SDC TM 4553/000/000

6 B ARDEN J A HAMILTON
Study of programming language effectiveness
University of Michigan Report #0322-1-F

7 P WASZKIEWICZ
BMD object code efficiency study
Final Report for Contract DAHC 60-70-C-048 dated 20 March 1970 Westinghouse Electric Corp

8 B N DICKMAN
BTL Private Communication

A review of recursive filtering algorithms

by BERNARD FRIEDLAND

The Singer Company
Little Falls, New Jersey

INTRODUCTION

The recursive filtering theory introduced scarcely a decade ago by Kalman[1,2] and Kalman and Bucy[3] has been widely hailed as a major development in data processing, perhaps as important as the work of Weiner[4] on linear filtering.

The introduction of Kalman's filtering theory could not have come at a more propitious time. The theory was expressed in the vector-matrix notation which had only recently been introduced to control engineers, and problems to which the theory was applicable were known to many investigators. Perhaps the most important reason for the almost immediate reception of the theory is that it is expressed essentially in the form of a computer program—in other words, it is naturally suited to implementation on a high-speed digital computer. Digital-computer implementation of earlier filtering and smoothing algorithms, on the other hand, did not seem to be very promising.

The fundamental result of Kalman's first papers can be stated fairly concisely: consider the discrete-time dynamic process evolving according to the "transition equation"

$$x_{n+1} = \Phi_n x_n + \Gamma_n u_n \tag{1.1}$$

where

- x_n is the "state vector" of the process at the nth instant of observation
- u_n is a vector of random excitations
- Φ_n is the (known) "transition matrix"
- Γ_n is a known matrix

Suppose that noisy observations y_n of the state are made in accordance with

$$y_n = H_n x_n + v_n \tag{1.2}$$

and, further, that u_n and v_n are gaussian, independent of each other and of u_k and v_k for all $k \neq n$. The *optimum estimate* $\hat{x}_n$ of x_n, given $y_n, y_{n-1}, \ldots$ can be expressed by means of the recursion equations

$$\hat{x}_n = \tilde{x}_n + K_n(y_n - H\tilde{x}_n) \tag{1.3}$$

$$\tilde{x}_n = \Phi_{n-1}\hat{x}_{n-1} \tag{1.4}$$

where

$$K_n = \tilde{P}_n H_n'(H_n \tilde{P}_n H_n' + R_n)^{-1} \tag{1.5}$$

$$\tilde{P}_n = \Phi_{n-1}\hat{P}_{n-1}\Phi_{n-1}' + \Gamma_{n-1}Q_{n-1}\Gamma_{n-1}' \tag{1.6}$$

$$\hat{P}_n = (I - K_n H_n)\tilde{P}_n \tag{1.7}$$

in which (′) denotes matrix transposition,

Q_n is the covariance matrix of the excitation noise u_n, i.e.,

$$Q_n = E[u_n u_n']$$

and

R_n is the covariance matrix of the observation noise.

The recursive nature of the Kalman filtering algorithm and its immediate translation into a computer program is illustrated in the flow chart of Figure 1a.

The reader may have noticed that the sense in which the estimate, computed by the algorithm defined by (1.3)-(1.6), is optimum has not yet been defined. This omission is intentional, because, quite remarkably, the estimate is optimum in every reasonable sense: it is the least-squares estimate, the minimum-variance estimate, and a maximum-likelihood estimate. It is also the conditional mean given the observation data, i.e.,

$$\hat{x}_n = E[x_n \mid y_n, y_{n-1}, \ldots]$$

The other quantities in (1.3)-(1.7) also have statistical

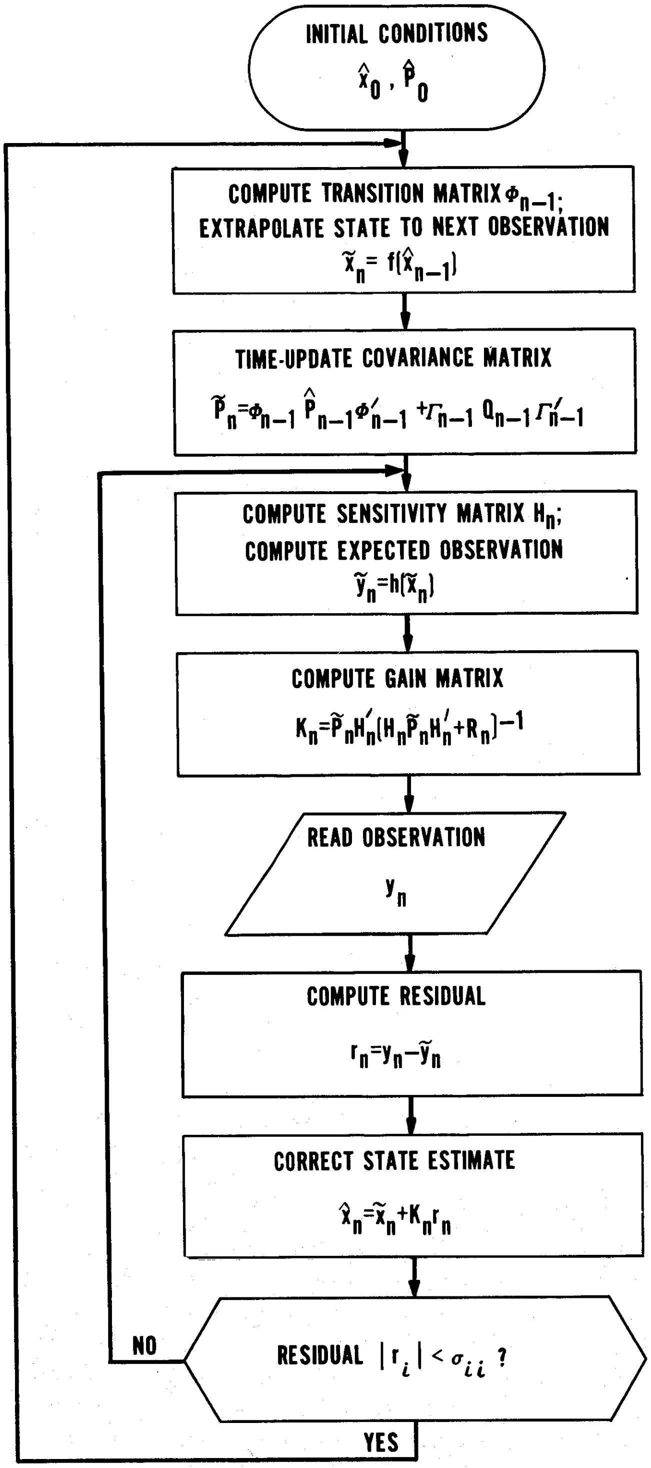

INITIAL CONDITIONS
$\hat{x}_0, \hat{P}_0$
COMPUTE TRANSITION MATRIX Φ_{n-1};
EXTRAPOLATE STATE TO NEXT OBSERVATION
$\tilde{x}_n = f(\hat{x}_{n-1})$
TIME-UPDATE COVARIANCE MATRIX
$\tilde{P}_n = \Phi_{n-1} \hat{P}_{n-1} \Phi'_{n-1} + \Gamma_{n-1} Q_{n-1} \Gamma'_{n-1}$
COMPUTE SENSITIVITY MATRIX H_n;
COMPUTE EXPECTED OBSERVATION
$\tilde{y}_n = h(\tilde{x}_n)$
COMPUTE GAIN MATRIX
$K_n = \tilde{P}_n H'_n (H_n \tilde{P}_n H'_n + R_n)^{-1}$
READ OBSERVATION
y_n
COMPUTE RESIDUAL
$r_n = y_n - \tilde{y}_n$
CORRECT STATE ESTIMATE
$\hat{x}_n = \tilde{x}_n + K_n r_n$
NO
RESIDUAL $|r_i| < \sigma_{ii}$?
YES

Figure 1a

significance, namely:

$\tilde{x}_n = E[x_n \mid y_{n-1}, y_{n-2}, \ldots]$, conditional mean of x_n, prior to observation y_n

$\hat{P}_n = E[(x_n - \hat{x}_n)(x_n - \hat{x}_n)']$, conditional a posteriori covariance matrix

$\tilde{P}_n = E[(x_n - \tilde{x}_n)(x_n - \tilde{x}_n)']$, conditional a priori covariance matrix

It is noted that the covariance matrices $\hat{P}_n$ and $\tilde{P}_n$ and hence the gain matrices K_n are independent of the observation data. Hence the latter can be computed off-line and stored for on-line implementation of (1.3) as indicated in the flow chart of Figure 1b. When u_n and v_n are correlated, Kalman's recursive filtering algorithm is somewhat more complicated and uses the covariance matrix $C_n = E[u_n v_n']$. The more general algorithm is given in References 1 and 2.

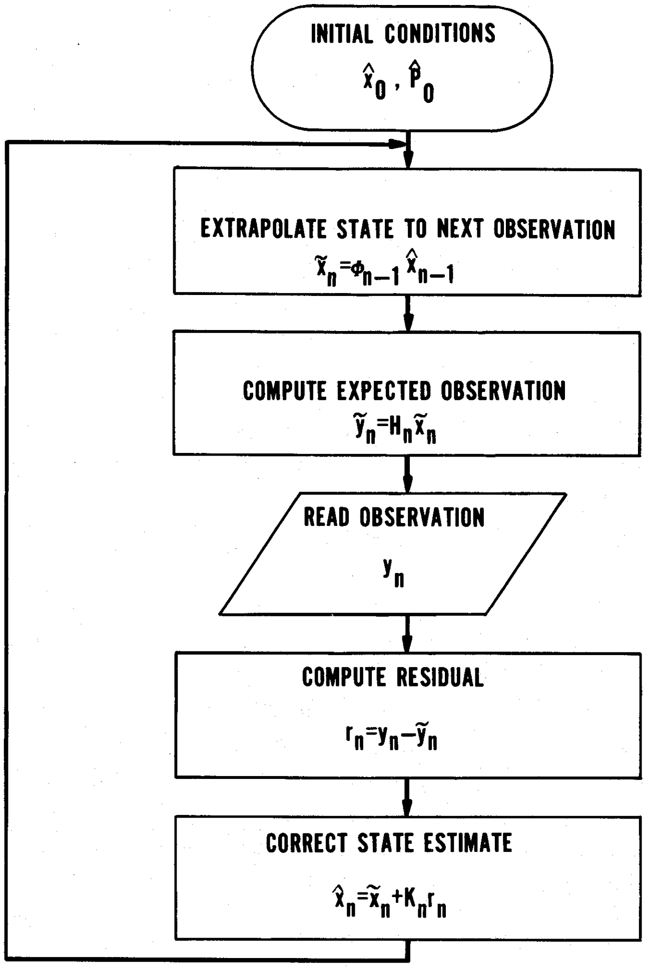

INITIAL CONDITIONS
$\hat{x}_0, \hat{P}_0$
EXTRAPOLATE STATE TO NEXT OBSERVATION
$\tilde{x}_n = \Phi_{n-1} \hat{x}_{n-1}$
COMPUTE EXPECTED OBSERVATION
$\tilde{y}_n = H_n \tilde{x}_n$
READ OBSERVATION
y_n
COMPUTE RESIDUAL
$r_n = y_n - \tilde{y}_n$
CORRECT STATE ESTIMATE
$\hat{x}_n = \tilde{x}_n + K_n r_n$

Figure 1b

Although recursive filtering is informally called "Kalman filtering," there is considerable dispute among the cognoscenti over who really deserves to be credited with the actual discovery. It would appear that the essentials of the theory were "in the wind" for several years prior to the appearance of Kalman's original paper and hence may have been discovered independently and published internally by several investigators just prior to the presentation of Kalman's paper at the 1959 Joint Automatic Control Conference. Earliest actual publication in a technical journal is claimed by Swerling,[5] and it is generally agreed that this is the earliest appearance of this theory.

DEVELOPMENT OF THE THEORY

The appearance of a new analytical technique of general utility is likely to engender a large number and variety of other studies. Recursive filtering is no exception: hundreds of papers and several textbooks have been written on this subject. Theoretical papers which have appeared since the original work of Kalman and Bucy treat with one or more of the following subjects.

- Alternative developments of the basic results and relationships with other subjects.
- Extensions to nonlinear and/or nongaussian processes.
- Subsidiary analysis of (usually messy) details.

Alternate developments of the theory

The fact that the estimate produced by the recursive filtering algorithm is optimum in many ways suggests also that there are many ways in which the same result can be derived. This is the case precisely: there seems to be as many different ways of deriving the results as there are users thereof; one ventures to say that every user has derived the results for himself at one time or another.

Kalman's original derivation makes use of a theorem on orthogonal projections for gaussian random variables and in the opinion of many investigators was unduly complicated. In the years that followed, derivations appeared which did not require all the mathematical apparatus used by Kalman.

The essential features of the recursive filtering algorithm can be developed without recourse to statistical concepts: the standard method of least squares and some ordinary matrix algebra are all that are needed to establish the basic relations. This development follows the line of reasoning suggested by Swerling[5] and is well-known to workers in the field. Because the derivation is fairly simple and will be useful for further discussion, we will present it here.

Suppose that the random excitations u_n on the process (1.1) are absent and it is required to determine the initial state vector x_0 on the basis of observation vectors $y_0, y_1, \ldots, y_n$. Using (1.1) and (1.2) it is seen that

$$\begin{aligned} y_n &= H_n\Phi_{n-1}\ldots\Phi_1 x_0 + v_n \\ &= H_n F_{n-1} x_0 + v_n \\ y_{n-1} &= H_{n-1}\Phi_{n-2}\ldots\Phi_2 x_0 + v_{n-1} \\ &= H_{n-1}F_{n-2}x_0 + v_{n-1} \\ &\vdots \\ y_0 &= H_0 x_0 + v_0 \end{aligned} \tag{2.1}$$

where $F_n = \Phi_n \ldots \Phi_1$. These equations can be concatenated into the single matrix equation

$$Y_n = M_n x_0 + V_n \tag{2.2}$$

where

$$Y_n = \begin{bmatrix} y_n \\ \vdots \\ y_0 \end{bmatrix} \quad M_n = \begin{bmatrix} H_n F_{n-1} \\ \vdots \\ H_0 \end{bmatrix} \quad V_n = \begin{bmatrix} v_n \\ \vdots \\ v_0 \end{bmatrix}$$

Suppose that for n sufficiently large, there is enough data to determine x_0. Then (2.2) will have a unique solution if there were no measurement errors. If there are more equations in (2.2) than components in the initial state, then (2.2) can be solved by means of weighted least squares. It is well-known that to minimize the quadratic form

$$Q = (Y_n - \bar{Y}_n)'W_n(Y_n - \bar{Y}_n) = V_n'W_nV_n$$

where Y_n is the observed sequence, $\bar{Y}_n$ is the sequence which would be observed in the absence of errors, and W_n is a (positive-definite, but otherwise arbitrary) weighting matrix, the optimum solution to (2.2) is

$$x_0^{(n)} = (M_n'W_nM_n)^{-1}M_n'W_nY_n \tag{2.3}$$

The superscript (n) on x_0 indicates that this is the initial state estimated on the basis of n measurements, and (2.3) is the standard, nonrecursive least squares formula. If the number of observations is large, the matrix M_n is correspondingly large: hence a fairly extensive calculation may be required to evaluate (2.3). Moreover, the calculation must be repeated each time a new observation or group of observations is to be included. Wouldn't it be convenient if the result of calculating the $(n-1)$st estimate $x_0^{(n-1)}$ could be used to simplify the calculation of $x_0^{(n)}$? To do this, the matrices M_n and W_n are partitioned as follows:

$$W_n=\left[\begin{array}{c|c} R_n^{-1} & 0 \\ \hline 0 & W_{n-1} \end{array}\right] \qquad M_n=\left[\begin{array}{c} H_nF_{n-1} \\ \hline M_{n-1} \end{array}\right] \tag{2.4}$$

(Note that this implies that the weighting matrix W_n consists of diagonal blocks. R_n^{-1} represents the new matrix in W_n that was not in W_{n-1}.) Using the submatrices in (2.4) it is found that (2.3) becomes

$$(F_{n-1}'H_n'R_n^{-1}H_nF_{n-1}+M_{n-1}'W_{n-1}M_{n-1})x_0^{(n)}$$
$$=F_{n-1}'H_n'R_n^{-1}y_n+M_{n-1}'W_{n-1}Y_{n-1} \tag{2.5}$$

where

$$Y_{n-1}=\begin{bmatrix} y_{n-1} \\ \vdots \\ y_0 \end{bmatrix}$$

Note, however, that if $x_0^{(n-1)}$ were computed by least squares, it would satisfy

$$Y_{n-1}=M_{n-1}x_0^{(n-1)}$$

Accordingly (2.5) becomes

$$(F_{n-1}'H_n'R_n^{-1}H_nF_{n-1}+M_{n-1}'W_{n-1}M_{n-1})x_0^{(n)}$$
$$=F_{n-1}'H_n'R_n^{-1}y_n+M_{n-1}'W_{n-1}M_{n-1}x_0^{(n-1)} \tag{2.6}$$

Note that (2.6) is already in recursive form: $x_0^{(n)}$ is expressed in terms of $x_0^{(n-1)}$ and the new observation y_n. To put the result of (2.6) in the form given earlier, we define a matrix $\tilde{P}_n$ by

$$F_{n-1}'\tilde{P}_n^{-1}F_{n-1}=M_{n-1}'W_{n-1}M_{n-1} \tag{2.7}$$

then (2.6) becomes

$$F_{n-1}x_0^{(n)}$$
$$=(\tilde{P}_n^{-1}+H_n'R_n^{-1}H_n)^{-1}(\tilde{P}_nF_{n-1}x_0^{(n-1)}+H_n'R_n^{-1}y_n) \tag{2.8}$$

after multiplication by the inverse of F_{n-1}' (which exists since F_n is the product of nonsingular transition matrices). It is also reasonable to define

$\tilde{x}_n=F_{n-1}x_0^{(n-1)}$, estimate of x_n using $x_0^{(n-1)}$

$\hat{x}_n=F_{n-1}x_0^{(n)}$, estimate of x_n using $x_0^{(n)}$

then (2.8) becomes

$$\hat{x}_n=({}_n\tilde{P}^{-1}+H_n'R_n^{-1}H_n)^{-1}[\tilde{P}_n^{-1}\tilde{x}_n+H_n'R_n^{-1}y_n] \tag{2.9}$$

Finally, we make use of a well-known matrix identity

$$(P^{-1}+H'R^{-1}H)^{-1}=P-PH'(HPH'+R)^{-1}HP \tag{2.10}$$

for arbitrary nonsingular matrices P and R. This reduces (2.9) to

$$\hat{x}_n=\tilde{x}_n+K_n(y_n-H_n\tilde{x}_n)$$

with

$$K_n=\tilde{P}_nH_n'(H_n\tilde{P}_nH_n'+R_n)^{-1}$$

which is the same as (1.3) and (1.5). Also, using (2.7) and (2.4)

$$F_n'\tilde{P}_{n+1}^{-1}F_n=M_n'W_nM_n$$
$$=F_{n-1}'H_n'R_n^{-1}H_nF_{n-1}+F_{n-1}'P_n^{-1}F_{n-1}$$
$$=F_{n-1}'[H_n'R_n^{-1}H_n+\tilde{P}_n^{-1}]F_{n-1} \tag{2.11}$$

But, $F_n=\Phi_nF_{n-1}$. Hence (2.11) becomes

$$\Phi_n'\tilde{P}_{n+1}^{-1}\Phi_n=\tilde{P}_n^{-1}+H_n'R_n^{-1}H_n \tag{2.12}$$

Let

$$\hat{P}_n^{-1}=\Phi_n'\tilde{P}_{n+1}^{-1}\Phi_n \tag{2.13}$$

Then, on using (2.10), (2.12) becomes

$$\hat{P}_n=(I-K_nH_n)\tilde{P}_n \tag{2.14}$$

Also, (2.13) becomes

$$\tilde{P}_{n+1}=\Phi_n\hat{P}_n\Phi_n' \tag{2.15}$$

It is seen that (2.14) is the same as (1.7) and (2.15) is the same as (1.6) with $Q_{n-1}\equiv 0$, which is equivalent to the absence of random excitation.

The derivation presented above emphasizes that recursive filtering is equivalent to weighted least squares when there is no random excitation on the process. The theoretical equivalence of the two methods, however, does not mean that there is nothing new in the recursive filtering algorithm. For even though the two methods are theoretically equivalent, they are computationally quite different. In particular, the standard least squares algorithm (2.3) entails inversion of a matrix whose dimensions are equal to the number of components in the state vector; the recursive algorithm, on the other hand, requires inversion of a matrix no larger than the number of simultaneous observations, i.e., the dimension of the observation vector. Since the latter, in most practical cases, is much smaller than the former, the recursive filtering algorithm generally permits computations to proceed with smaller matrices. This feature, however, is not the most significant. More significant is the numerical problem which arises when the number of observations is large. In this case the matrix $M_n'W_nM_n$, which is to be inverted for the least squares estimate, has long, skinny matrices M_n' and M_n as its outer factors. Hence a single element in $M_n'W_nM_n$ could be the sum of a large number of products. If the individual products are comparable in magnitude and could have different signs, it is evident that the accumulated round-off error could be enough to make the elements of the resulting matrix a set of random numbers. All following calculations would then be meaningless. Since the recursive calculation never entails dealing directly with large matrices, computational problems are considerably forestalled—but not entirely eliminated.

One might think that the equivalence of the solution to a statistical problem—finding the conditional mean of a random process—is equivalent to the solution of nonstatistical problem is a fortuitous circumstance. This, however, is not the case. It turns out that this equivalence applies in a more general context as discussed in the recent, excellent book by Jazwinski.[6]

The equivalence of the conditional mean, the minimum variance estimate, and the maximum likelihood estimate, has led to a variety of derivations based on statistical concepts, in addition to the orthogonal projection theorem used by Kalman. For example, the results of the first section can be derived by use of Bayes' Theorem and/or by finding the expression for the probability density function for x_n, given $y_n, \ldots y_0$.

Continuous-time processes

The problem originally considered by Kalman concerned a discrete-time dynamic process (1.1) with discrete-time ("sampled-data") observations. The extension to a continuous-time process without random excitation, and with sampled observations, is immediate. Suppose the process is governed by

$$\dot{x}=A(t)x \tag{2.16}$$

with observations taken at discrete-times

$$y_n=y(t_n)=H(t_n)x(t_n)+v_n$$

The solution to (2.16) can be expressed by

$$x(t_{n+1})=\Phi(t_{n+1},t_n)x(t_n)$$

where $\Phi(t_{n+1}, t_n)$ is the transition matrix, which is the solution at time t_{n+1} to the matrix differential equation

$$\dot{\Phi}=A(t)\Phi \tag{2.17}$$

with $\Phi(t_n)=I$, the identity matrix. Obviously this transition matrix is used in place of Φ_n, and the earlier discrete-time results are directly applicable.

The transition matrix Φ_n is needed in two places: to update the state estimate via (1.4) and to update the covariance matrix via (1.6). It is worth noting, however, that the transition matrix is not needed to update the state estimate. Instead the differential equation (2.16) can be integrated over the interval $[t_n, t_{n+1}]$ with $x(t_n)=\hat{x}_n$ to obtain $x(t_{n+1})=\tilde{x}_{n+1})$. This additional computation may be justified when it is believed that the evaluation of the transition matrix may be inaccurate, because inaccuracies in the transition matrix in updating the covariance matrix (1.5) are far less serious than errors in the transition matrix used to evaluate the estimate of the state. The former type of error will result in using a somewhat erroneous gain matrix K_n, but the latter will result in a direct error in the estimate of the state.

When random excitation is present on the dynamics, a variety of theoretical problems arise the treatment of which entails sophisticated mathematical concepts. Jazwinski's book[6] contains a very well written account of the relevant mathematical techniques. The major difficulty arises because the continuous time analog of the random excitation v_n is white noise which has very disagreeable mathematical properties. Consider the effect of adding white noise to the process (2.16), i.e.,

$$\dot{x}=A(t)x+B(t)\xi \tag{2.18}$$

with ξ being white noise having a known spectral density matrix. If ξ were nonrandom, the solution to (2.18) would be

$$x(t_{n+1})=\Phi(t_{n+1},t_n)x(t_n)+\int_{t_n}^{t_{n+1}}\Phi(t_{n+1},t)B(t)\xi(t)\,dt \tag{2.19}$$

Since $\xi(t)$ is white noise, however, the integral in (2.19) must be interpreted in a stochastic sense. In accordance with standard theory, the integral

$$v_n=\int_{t_n}^{t_{n+1}}\Phi(t_{n+1},t)B(t)\xi(t)\,dt$$

is a random variable, uncorrelated with v_k, $k\neq n$ having zero mean and covariance matrix:

$$Q_n=\int_{t_n}^{t_{n+1}}\Phi(t_{n+1},t)B(t)\Sigma(t)B'(t)\Phi'(t_{n+1},t)\,dt \tag{2.20}$$

where $\Sigma(t)$ is the spectral density matrix of the white noise $\xi(t)$, i.e., $E[\xi(t)\xi'(\tau)]=\Sigma(t)\delta(t-\tau)$. Hence, except for the possible difficulty in evaluating (2.20), the discrete-time theory is applicable to (2.19).

When the observations are continuous, however, i.e.,

$$y(t)=H(t)x(t)+\eta(t) \tag{2.21}$$

with $\eta(t)$ being white noise, the discrete-time algorithm is not directly applicable. A heuristic deviation of the continuous-time equations of Kalman filter is possible by use of a limiting process as described by Kalman.[2] For small intervals of time, the solution to (2.18) can be approximated by

$$x(t_{n+1})=[I+A(t_n)\Delta t_n]x(t_n)+B(t_n)v_n \tag{2.22}$$

where

$$\Delta t=t_{n+1}-t_n$$

v_n is a random variable of zero mean and variance $\Sigma(t_n)\Delta t_n$, owing to the integration of white noise for an interval Δt_n.

Owing to the white noise on the observations it is not possible to sample the output $y(t)$ at discrete times; instead, it is necessary to smooth the output during the intervals. Simple averaging is easiest to treat. For this type of smoothing

$$y_n=\frac{1}{\Delta t}\int_{t_{n-1}}^{t_n} y(t)\,dt \doteq y(t_n)+w_n=H(t_n)x(t_n)+w_n \tag{2.23}$$

where

$$w_n=\frac{1}{\Delta t}\int_{t_{n-1}}^{t_n}\eta(t)\,dt$$

The integral in w_n is a random variable of zero mean and variance $N\Delta t$ where N is the spectral density matrix of the white noise η. Accordingly

w_n is a random variable of zero mean and variance

$$(1/\Delta t)^2 N\Delta t=N/\Delta t$$

Using (2.22) and (2.23) as the characterization of an approximate discrete-time system, and applying the discrete-time filtering algorithm, gives

$$\hat{x}(t_{n+1})=[I+A(t_n)\Delta t]\{\hat{x}(t_n)+\hat{P}_nH'(t_n)R_n^{-1}[y_n-H(t_n)\hat{x}(t_n)]\}$$

Substituting $R_n=N/\Delta t$ into this equation and rearranging terms gives

$$\frac{\hat{x}(t_{n+1})-\hat{x}(t_n)}{\Delta t}=[A(t_n)\hat{x}(t_n)+\hat{P}_nH'(t_n)N^{-1}(y_n-H(t_n)\hat{x}(t_n)]+O(\Delta t)$$

Hence, as $\Delta t\to 0$, we obtain

$$\dot{\hat{x}}=A(t)\hat{x}+K(y-H(t)\hat{x}) \tag{2.24}$$

with

$$K=\hat{P}_nH'(t)N^{-1}$$

Also, the variance equations (1.6)-(1.7), which can be written in the form

$$\hat{P}_{n+1}^{-1}=(\Phi_n\hat{P}_n\Phi_n'+\Gamma_nQ_n\Gamma_n)^{-1}+H_{n+1}'R_{n+1}H_{n+1}$$

become

$$\hat{P}_{n+1}^{-1}=\{\hat{P}_n+[A(t_n)P_n+P_nA'(t_n)+B(t_n)\Sigma(t_n)B'(t_n)]\Delta t+O(\overline{\Delta t}^2)\}^{-1}+H'(t_n)N^{-1}H(t_n)\Delta t$$

Upon rearranging terms and omitting terms of $O(\overline{\Delta t}^2)$ we obtain

$$\frac{\hat{P}_{n+1}-\hat{P}_n}{\Delta t}=A(t_n)P_n+P_nA'(t_n)+B(t_n)\Sigma(t_n)B'(t_n)-P_nH'(t_n)N^{-1}H(t_n)P_n$$

Finally, passing to the limit as $\Delta t\to 0$, we get

$$\dot{\hat{P}}=A(t)\hat{P}+\hat{P}A'(t)+B(t)\Sigma(t)B'(t)-\hat{P}H'(t)N^{-1}H(t)\hat{P} \tag{2.25}$$

Thus the optimum state estimate evolves in accordance with the differential equation of the original process, but forced by the term $K(y-H\hat{x})$. The covariance matrix P evolves in accordance with the matrix Riccati equation (2.25).

Rigorous treatment of the continuous-time process entails rather sophisticated concepts of stochastic processes. The original treatment by Kalman and Bucy[3] is entirely rigorous, but is confined to linear systems. More general formulations which, in principle, are capable of generalization to nonlinear processes, have been developed by Stratonovich,[7] Kushner,[8,9] Bucy,[10] and Kailath and Frost,[11,12] and by others.

EXTENSION OF THE THEORY

Like most theories, the theory of recursive filtering is applicable exactly in almost no practical situation: few processes are linear, almost no random noise is gaussian (although what it is if not gaussian is another problem), and frequently the actual process model is not known. All these problems have received attention in the decade since the appearance of the Kalman-Bucy papers, and are still receiving attention at the present time. A comprehensive survey of the various extensions which have been published is an overwhelming assignment—a comprehensive bibliography could easily contain a thousand items. There are, however, several trends in the investigations being pursued which can be outlined in broad perspective.

Nonlinear discrete-time processes

A general discrete-time process could be represented by the difference equation

$$x_{n+1}=f(x_n, u_n) \tag{3.1}$$

a general observation may be expressed by

$$y_n=h(x_n, v_n) \tag{3.2}$$

where, in most u_n and v_n are random variables about which very little is known. The usual way of handling a nonlinear problem is to linearize it along a nominal solution. In the present case, suppose a nominal sequence $\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_n$ were known, and, moreover, it is known *a priori* that the random quantities u_n and v_n are "small enough" to insure that the differences

$$e_n=x_n-\bar{x}_n \tag{3.3}$$

are small. Then (3.1) and (3.2) become

$$\begin{aligned} x_{n+1} &\doteq f(\bar{x}_n)+\bar{\Phi}_n(x_n-\bar{x}_n)+\bar{\Gamma}_n u_n \\ y_n &= h(\bar{x}_n)+\bar{H}_n(x_n-\bar{x}_n)+\bar{Z}_n v_n \end{aligned} \tag{3.4}$$

but

$$\bar{x}_{n+1}=f(\bar{x}_n)$$

$$y_n=h(\bar{x}_n)= \text{nominal observation}$$

where

$$\bar{\Phi}_n=\left[\frac{\partial f(x_n, 0)}{\partial x_n}\right]_{x_n=\bar{x}_n}$$

$$\bar{\Gamma}_n=\left[\frac{\partial f(x_n, 0)}{\partial u_n}\right]_{x_n=\bar{x}_n}$$

$$\bar{H}_n=\left[\frac{\partial h(x_n, 0)}{\partial x_n}\right]_{x_n=\bar{x}_n}$$

$$\bar{Z}_n=\left[\frac{\partial h(x_n, 0)}{\partial x_n}\right]_{x_n=\bar{x}_n}$$

or

$$e_{n+1}=\bar{\Phi}_n e_n+\bar{\Gamma}_n u_n+O(e_n^2) \tag{3.5}$$

$$y_n-\bar{y}_n=\bar{H}_n e_n+\bar{Z}_n v_n+O(e_n^2) \tag{3.6}$$

Since (3.5) and (3.6), after omitting the terms of $O(e_n^2)$ constitute a linear system, the linear theory of the previous section is applicable, and the deviations from the nominal trajectory are given by

$$\tilde{e}_{n+1}=\bar{\Phi}_n\hat{e}_n \tag{3.7}$$

$$\hat{e}_n=\tilde{e}_n+K_n(y_n-H_n\tilde{e}_n) \tag{3.8}$$

or in terms of the total state

$$\begin{aligned} \tilde{x}_{n+1} &= f(\bar{x}_n)+\bar{\Phi}_n\hat{e}_n \\ &= f(\hat{x}_n)-\hat{\Phi}_n(\hat{x}_n-\bar{x}_n)+\bar{\Phi}_n(\hat{x}_n-\bar{x}_n) \end{aligned} \tag{3.9}$$

$$\hat{\Phi}_n=\left[\frac{\partial f(x_n, 0)}{\partial x_n}\right]_{x_n=\hat{x}_n}$$

Note that except for the difference between $\bar{\Phi}_n$ and $\hat{\Phi}_n$ (3.9) can be written

$$\tilde{x}_{n+1}=f(\hat{x}_n) \tag{3.10}$$

Moreover, in many situations the estimated solutions $\hat{x}_n$ is more accurate than any *a priori* nominal solution $\bar{x}_n\neq\hat{x}_n$; in fact, there may be no reasonable way of determining $\bar{x}_n$ other than by making it identical to the $\hat{x}_n$, i.e., by *defining* the nominal solution as the running estimate. In this case (3.9) and (3.10) are identical. The same reasoning gives

$$\hat{x}_n=\tilde{x}_n+K_n[y_n-h(\tilde{x}_n)] \tag{3.11}$$

The recursive algorithm (3.10) and (3.11) with the gain matrix computed from (1.5) and (1.6) with

$$\Phi_n=\hat{\Phi}_n, \qquad H_n=\hat{H}_n$$

is the so-called "extended" Kalman filter, as it is now termed. These equations were used as early as 1962 by Smith.[13] The following features of the extended Kalman filter are worth emphasizing. See Figure 2.

- No nominal trajectory is required
- Covariance matrices $\tilde{P}_n$, $\hat{P}_n$, and gain matrix K_n must be computed on-line
- Statistical optimality is only approximate.

Although no nominal trajectory is required, it *is* necessary to compute the covariance matrix on-line and this computation occupies the bulk of the computation requirements. It is not insignificant that strict optimality cannot be assured for the extended Kalman filter; as a matter of fact, it often happens that the estimate given by (3.10) and (3.11) actually *diverges* from the true trajectory as the number of observations processed increases. This "divergence problem," to be discussed subsequently, could result because the terms of $O(e_n^2)$, so casually omitted in developing the filtering equations, in fact, may not be negligible. Their presence

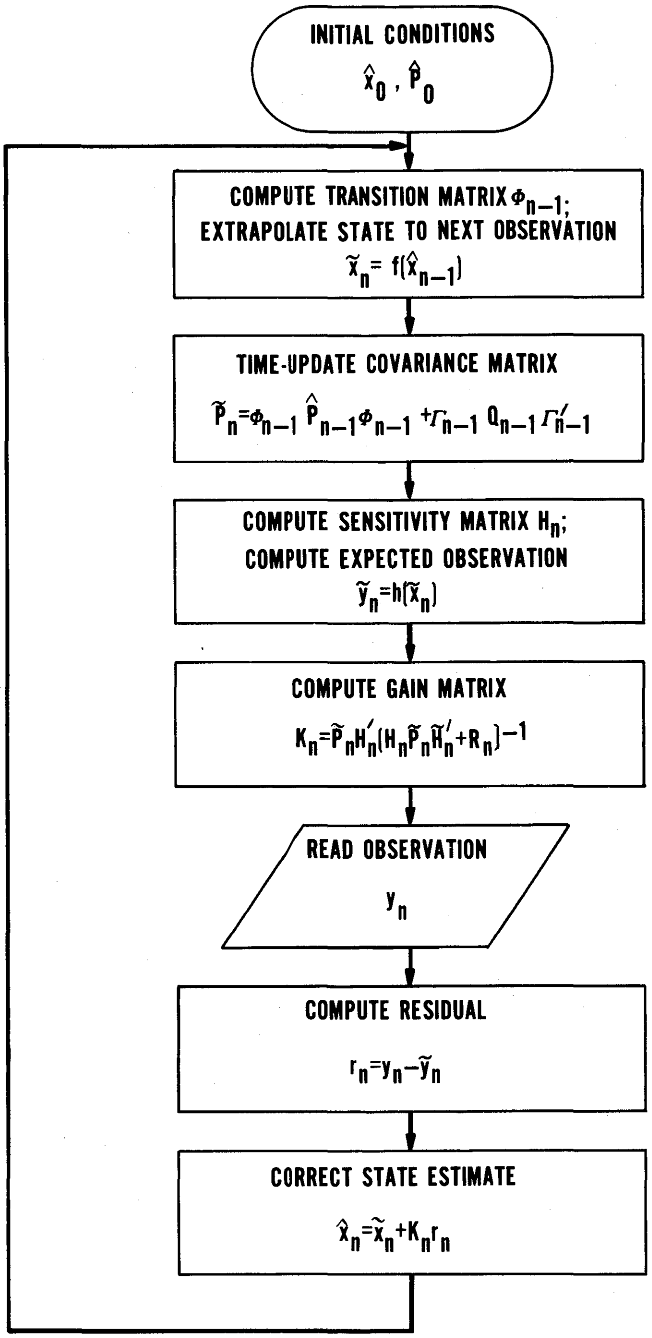

INITIAL CONDITIONS
$\hat{x}_0$, $\hat{P}_0$
COMPUTE TRANSITION MATRIX Φ_{n-1};
EXTRAPOLATE STATE TO NEXT OBSERVATION
$\tilde{x}_n = f(\hat{x}_{n-1})$
TIME-UPDATE COVARIANCE MATRIX
$\tilde{P}_n = \Phi_{n-1} \hat{P}_{n-1} \Phi_{n-1} + \Gamma_{n-1} Q_{n-1} \Gamma'_{n-1}$
COMPUTE SENSITIVITY MATRIX H_n;
COMPUTE EXPECTED OBSERVATION
$\tilde{y}_n = h(\tilde{x}_n)$
COMPUTE GAIN MATRIX
$K_n = \tilde{P}_n H'_n (H_n \tilde{P}_n \tilde{H}'_n + R_n)^{-1}$
READ OBSERVATION
y_n
COMPUTE RESIDUAL
$r_n = y_n - \tilde{y}_n$
CORRECT STATE ESTIMATE
$\hat{x}_n = \tilde{x}_n + K_n r_n$

Figure 2

can introduce a statistical bias into the results; the effect, paradoxically, is most serious when the random noise represented by $\bar{\Gamma}_n u_n$ and $\bar{Z}_n v_n$ in (3.5) and (3.6), respectively, is small relative to the nonlinear terms. When the random terms are large they tend to swamp the nonlinear terms; i.e., those of $O(e_n^2)$; when they are small the nonlinear terms may be predominate. Since the nonlinear terms add to the random terms, one is tempted to increase the covariance matrices of the latter in a heuristic attempt to account for nonlinearity. This approach is often quite successful, but could be dangerous because the nonlinear terms are not random, but are related to the linear terms in a systematic way.

The limitations of the linearized Kalman filtering equation (3.7) and (3.8) or the extended Kalman filter (3.10) and (3.11) have led to a number of investigations, some still continuing, of more accurate extensions of the basic theory or of alternate approaches. These investigations appear to have been motivated by two considerations: the desire to achieve performance closer to the optimum, and the need to avoid problems which arise in applying the linearized or extended algorithm.

The desire to avoid the "divergence" attributable to omission of nonlinear terms is a practical motivation which may be handled in various ways, not necessarily related to achieving "better" (i.e., closer to optimum) performance. In the presence of nonlinearities and/or nongaussian random variables, the very definition of the optimum estimate is a problem: in the linear, gaussian case the conditional mean is also the minimum variance estimate and the maximum likelihood (Bayes) estimate. In the linear, but nongaussian case, the minimum variance estimate (which is what is obtained by applying the Kalman-filtering algorithm) is not always the conditional mean, and neither may be the maximum likelihood estimate. In the nonlinear case, moreover, none of the estimates can be expressed by an exact recursion algorithm; errors are introduced by use of truncated approximations the effects of which are very difficult to assess. As a practical matter performance of nonlinear filters can be assessed only by Monte-Carlo stimulation: a "fix" which works well in one application may be useless or even worse than the basic extended algorithm in another.

Improved estimation algorithms can generally be classified in two categories: higher order, explicit algorithms and iterative algorithms. In the former, the updated estimate $\hat{x}_{n+1}$ is obtained from $\hat{x}_n$ by an explicit computation which entails no iteration and no check of accuracy; in the iterative techniques, the updated estimate $\hat{x}_{n+1}$ is an implicit function of $\hat{x}_n$ which is solved for iteratively.

Iterative techniques

Iterative techniques for treatment of nonlinear problems may be visualized as generalizations of the ancient Newton-Raphson technique for solving nonlinear equations. Suppose, for example, that we have a single observation y from which it is possible to determine the state x through the solution of the system of equations represented by

$$y = h(x) \tag{3.12}$$

If this is true, the Jacobian matrix

$$H(x) = \left[\frac{\partial h}{\partial x}\right]$$

is nonsingular in the vicinity of the solution to (3.12). The standard Newton-Raphson iteration algorithm for solving (3.12) is

$$\hat{x} = \tilde{x} + [H(\tilde{x})]^{-1}[y - h(\tilde{x})] \tag{3.13}$$

where

$\tilde{x}$ is a prior estimate of the solution

$\hat{x}$ is a revised estimate

When $\hat{x}$ differs significantly from $\tilde{x}$, $\tilde{x}$ is replaced by $\hat{x}$ and the iteration is continued until $|| \hat{x} - \tilde{x} ||$ is sufficiently small in accordance with some criterion established by the user.

Comparison of (3.13) with the observation update equation (3.11) of extended Kalman filtering reveals a very striking similarity. In Kalman filtering the gain matrix

$$K_n = \tilde{P}_n H_n'(H_n \tilde{P}_n H_n' + R_n)^{-1}, \quad H_n = H(\tilde{x}_n)$$

multiplies the residual $y - h(\tilde{x})$ while in the Newton-Raphson algorithm the residual is multiplied by H_n^{-1}. If the observation noise is absent, and H_n^{-1} exists, however, the Kalman filter gain becomes

$$K_n = \tilde{P}_n H_n'(H_n')^{-1} \tilde{P}_n^{-1} H_n^{-1} = H_n^{-1}$$

In other words the Kalman filter gain is the same as the Newton-Raphson weighting matrix when the latter exists and the noise is negligible. The Kalman gain matrix may thus be regarded as a generalized version of the Newton-Raphson gain which reduces to the latter in the case when noise is absent and H_n^{-1} exists. When H_n^{-1} does not exist, i.e., when the number of observations is insufficient to permit determination of the state, but noise is absent, then the Kalman gain matrix $K_n = \tilde{P}_n H_n'(H_n \tilde{P}_n H_n')^{-1}$ is a "generalized inverse" of H_n in the sense that $H_n K_n = I$, regardless of the value of $\tilde{P}_n$.

Although the Kalman filter uses a more general gain matrix, it does not perform the iterations of the Newton-Raphson technique. In the linear case, of course, the iterations are not necessary. In the nonlinear case, however, one would expect that the initial estimate $\tilde{x}$ of the state could be improved upon by iteration unless the uncertainty in the observation has an effect equal to the effect of the nonlinearity. Thus, in an application entailing highly accurate, but nonlinear, sensors, one could expect significant benefit from use of the iterations; conversely, when the sensor noise is large relative to the nonlinearity, little improvement due to the iteration is to be expected.

Assuming that iterations are to be employed, what is a suitable convergence criterion? If (3.12) is solved exactly—to the precision available to the computer—then all past history of the process is ignored. If no iterations are used, then nonlinear effects are disregarded. A reasonable convergence criterion would be to iterate until the contribution of the error to the residual vector $y - h(\tilde{x})$ is just about as large as the nonlinear effect. The (approximate) covariance matrix of the residual is $H_n \tilde{P}_n H_n' + R_n$. Since, in the absense of nonlinear effects, most of the residuals can be expected to be less than their corresponding theoretical standard deviation, i.e.,

$$[y_i - h_i(\tilde{x})]^2 < [H_n \tilde{P}_n H_n' + R_n]_{ii}^2 = \sigma_i^2 \tag{3.14}$$

it would be reasonable to continue iteration until (3.14) is satisfied. The addition of the Newton-Raphson iteration cycle to account for nonlinear observations entails a negligible complication of the basic algorithm, as the flow-chart of Figure 3 shows. The operating time, of course, will be increased in proportion to the average number of iterations necessary to pass the convergence test. Whether this additional running time is justified depends on the improvement achieved. Moreover, in an application in which the computer is dedicated to the Kalman filtering algorithm, there may be time available for the iteration so that the additional cost could be inconsequential.

The ideas inherent in the Newton-Raphson iteration technique may be extended to the case in which the state cannot be determined from a single observation, but requires a series of observations and knowledge of the process dynamics. The general equations may be derived, using considerations of least-squares by combining the analysis of the second section and of this section. The inherent statistical parameters can be brought into the problem formulation explicitly. Typical developments are to be found in Cox,[14] and Friedland and Bernstein.[15]

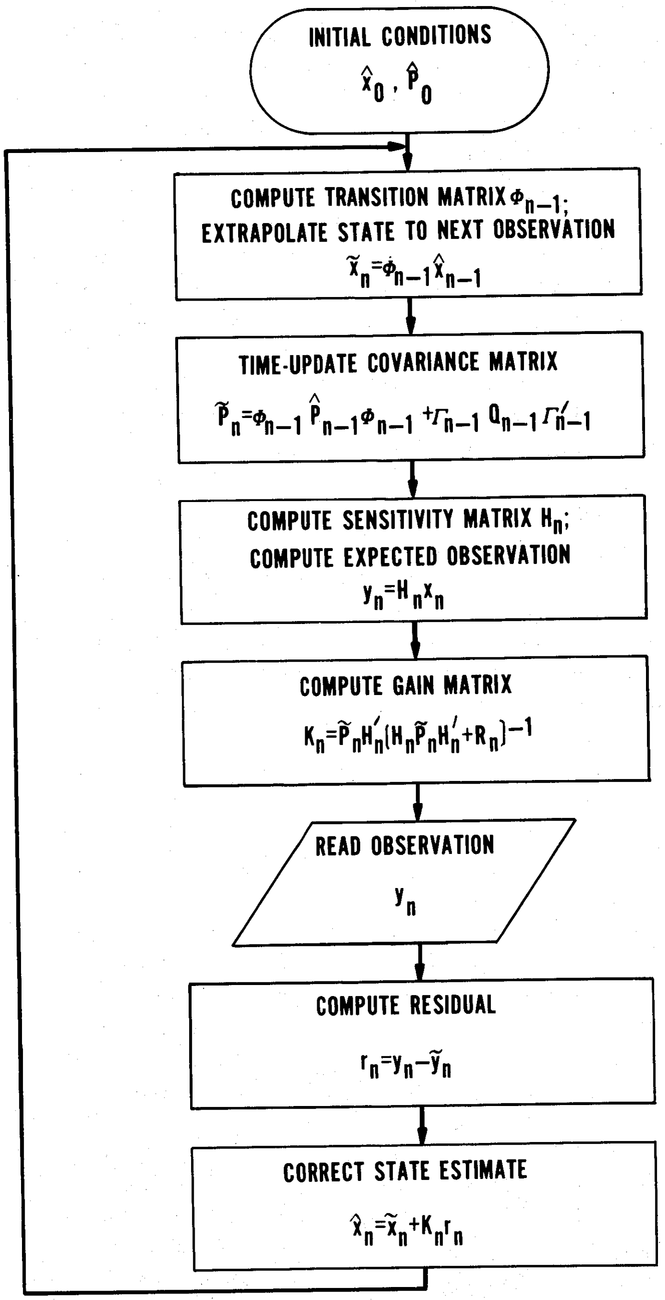

INITIAL CONDITIONS
$\hat{x}_0, \hat{P}_0$
COMPUTE TRANSITION MATRIX Φ_{n-1};
EXTRAPOLATE STATE TO NEXT OBSERVATION
$\tilde{x}_n=\Phi_{n-1}\hat{x}_{n-1}$
TIME-UPDATE COVARIANCE MATRIX
$\tilde{P}_n=\Phi_{n-1}\hat{P}_{n-1}\Phi_{n-1}+\Gamma_{n-1}Q_{n-1}\Gamma'_{n-1}$
COMPUTE SENSITIVITY MATRIX H_n;
COMPUTE EXPECTED OBSERVATION
$y_n=H_nx_n$
COMPUTE GAIN MATRIX
$K_n=\tilde{P}_nH'_n(H_n\tilde{P}_nH'_n+R_n)^{-1}$
READ OBSERVATION
y_n
COMPUTE RESIDUAL
$r_n=y_n-\tilde{y}_n$
CORRECT STATE ESTIMATE
$\hat{x}_n=\tilde{x}_n+K_nr_n$

Figure 3

Explicit techniques

The general advantage of iterative techniques is their relative simplicity; the potential disadvantages include uncertain convergence, and possible inefficiency in terms of running time. As an alternative to iterative techniques for nonlinear estimation, it is possible to consider explicit algorithms, i.e., those in which a fixed number of calculations are performed at each step.

The most obvious approach is based on the recognition that the extended Kalman filtering algorithm can be interpreted as the low-order terms in series expansions. Inclusion of additional terms could justifiably be expected to produce better results. Return again to the simplest case in which a single observation suffices to determine the state, i.e., $y=h(x)$ where $h(\cdot)$ is an invertible nonlinear function. In this case we could write, for arbitrary x

$$x=\varphi(y)$$

where $\varphi(x)$ is the inverse function of $h(\cdot)$. Assuming the existence of the required derivatives, one could write

$$x=\varphi(\tilde{y})+\left[\frac{\partial\varphi}{\partial\tilde{y}}\right](y-\tilde{y})+\frac{1}{2}\left[\frac{\partial^2\varphi}{\tilde{y}^2}\right](y-\tilde{y})^2+\cdots \quad (3.15)$$

where, in the multivariable case

$$\left[\frac{\partial\varphi}{\partial y}\right]=\left[\frac{\partial\varphi_i}{\partial y_j}\right], \quad \text{a matrix}$$

$$\left[\frac{\partial^2\varphi}{\partial y^2}\right]=\left[\frac{\partial^2\varphi_i}{\partial y_jy_k}\right], \text{ a third rank tensor.}$$

If

$$\tilde{y}=h(\tilde{x}), \qquad \tilde{x}=\varphi(\tilde{y})$$

Then (3.15) becomes

$$x=\tilde{x}+\left[\frac{\partial\varphi}{\partial y}\right][y-h(\tilde{x})]+\frac{1}{2}\left[\frac{\partial^2\varphi}{\partial y^2}\right][y-h(\tilde{x})]^2+\cdots \quad (3.16)$$

Moreover

$$\left[\frac{\partial\varphi}{\partial\tilde{y}}\right]=\left[\frac{\partial h}{\partial\tilde{x}}\right]^{-1}$$

Hence it is seen that the Newton-Raphson iteration formula (3.13) comprises the first two terms of a series. Also, the Kalman algorithm can be viewed as a statistical generalization of the Newton-Raphson formula. It is thus reasonable to expect that an explicit nonlinear filtering algorithm could be obtained as a generalization of (3.16), i.e.,

$$\hat{x}_n=\tilde{x}_n+K_n^{(2)}[y-h(\tilde{x}_n)]+K_n^{(3)}[y-h(\tilde{x}_n)]^2+\cdots \quad (3.17)$$

where $K_n^{(i)}$ denotes a tensor of rank i and the expression $K^{(i)}r^{i-1}$ denotes tensor multiplication, i.e., contraction to a vector. The use of an explicit formula of the form (3.17) raises several theoretical and practical questions.

The major theoretical question is the definition of an optimum estimate. Whereas in the linear, gaussian case all estimates are equivalent and thus lead to the same algorithm; this is not the case in the nonlinear case. As a consequence the weighting tensors $K_n^{(i)}$ in (3.17) depend on how the estimate is defined. It would be comforting to know that the differences between the weighting tensors for different estimates are insignificant. Unfortunately, this is not the case: even the sign of the second order term may depend on the definition employed.

The practical question is concerned with the computational requirements of explicit methods. It is clear that both storage and time requirements become very large to compute even the second-order terms in a system with a large-dimension state vector. To justify the additional computer requirements, it is necessary to show that use of higher-order terms really provide significant benefit in terms of performance. The reader interested in further pursuing higher order methods, as well as iterative methods, will find an excellent discussion and additional references in Chapters 8 and 9 of Jazwinski's book.[6]

The Kalman filtering technique for linear processes has a theoretical basis in the fact that equations for the propagation of the conditional mean $\hat{x}$ and conditional variance $\hat{P}$ are relatively simple and, moreover, serve to completely characterize the conditional probability distribution of the random state vector x given the observation data $y(t)$. In the nonlinear, nongaussian case, the conditional mean and covariance do not serve completely to characterize the conditional probability density function, and any finite number of moments can only yield an approximation to the actual probability density function. Since the conditional density function is all one needs to determine everything one would want to know about the variable, Bucy reasoned that there may be better methods of learning how the density function propagates than by propagating all its moments. In particular, he and Senne[16] developed a method which is completely different from the series methods. This method is based representing the density function by its actual numerical values at points in a high-dimensional grid and developing a method of representing the motion of this density function in space. Owing to the large number of points needed to represent a high-dimensional density function, only the points at which the density function is "concentrated" are determined and propagated. Although the method seems quite complicated, Bucy and Senne claim it is competitive with series-expansion methods and, in cases they have examined, has given better results than other methods.

Nonlinear continuous-time processes

Very few real processes start out with discrete-time models. In most cases a continuous-time model (in the form of differential equations) is the starting point. When there is no random excitation on the process, and observations are made at discrete instants of time, there is little difficulty in applying the discrete-time (extended-linear) theory. Suppose for example that the process is governed by

$$\dot{x} = \alpha(x) \tag{3.18}$$

The solution of (3.18):

$$x(t_n) = x(t_{n-1}) + \int_{t_{n-1}}^{t_n} \alpha[x(\lambda)]\, d\lambda \tag{3.19}$$

defines the function $f(x_n)$ in (3.1). Thus the time-update (3.10) is computed by numerical integration of (3.18) over the intervals between observations. The initial condition for starting the integration is $\hat{x}_{n-1} = \hat{x}(t_{n-1})$ and the result, just prior to the next observation at t_n is $\bar{x}(t_n)$. The update for the nth observation is then computed through use of (3.11).

In order to update the covariance matrix, in accordance with (1.6), it is necessary to have the differential transition matrix. Since an analytic expression for the nonlinear function f is not generally available, it is not possible to calculate $\Phi = [\partial f/\partial x]$ analytically. Owing to the definition of f as the solution of a system of ordinary differential equations, however, it is possible to compute the transition matrix by numerical integration of the matrix differential equation

$$\dot{\Phi} = \left[\frac{\partial \alpha}{\partial x}\right] \Phi \tag{3.20}$$

with the initial condition $\Phi(t_{n-1}) = I$, where $[\partial\alpha/\partial x]$ is the Jacobian matrix of $\alpha(x)$ evaluated along the solution to (3.18). Thus $x(t_n)$ and Φ_n are evaluated concurrently by numerical integration of a system of $n(n+1)$ differential equations. The process differential equations must be integrated to an accuracy consistent with the observation errors; in other words, the numerical errors in obtaining $\bar{x}_n = \bar{x}(t_n)$ from $\hat{x}(t_{n-1})$ must be of a lower order than the correction $K_n[y_n - h(\bar{x}_n)]$. The computation of the transition matrix, however, requires accuracy only consistent with the accuracy to which the statistical parameters are known. In practice this usually means that the numerical integration algorithm used for (3.20) can be less sophisticated than that used to integrate (3.18). For example, while a fourth-order Runge-Kutta algorithm might be needed to integrate (3.18), it could well happen that a simple, first-order Euler algorithm

$$\Phi_n = I + \left[\frac{\partial\alpha}{\partial x}\right] (t_n - t_{n-1})$$

would provide sufficient accuracy for the state transition matrix.

Nonlinear dynamics with random excitation create problems on the frontiers of investigation in the mathematical theory of stochastic processes. The very meaning of differential equations with white noise excitation is open to serious question: almost nothing is known about how to interpret a general differential equation of the form $\dot{x} = \alpha(x, \xi)$ with ξ being white noise; the meaning of the more restrictive equation

$$\dot{x} = \alpha(x) + B(x)\xi$$

in which the white noise ξ is additive, is open to two interpretations, depending upon whether one uses the mathematically-favored "Itô calculus" or the "Stratonovich calculus" unless the matrix B does not depend on x. A lucid discussion of the two interpretations is given by Jazwinski.[6] Since white noise is a mathematical abstraction which cannot exist in nature, the different interpretations are of more interest to mathematicians than to engineers and physicists. Moreover, unless a very precise analysis of the physical behavior of the dynamic process is made, it is usually impossible to establish the precise nature of the random excitation. For this reason it would seem reasonable to account for all the random effects in a typical process by a model of the form

$$x_{n+1} = f(x_n) + \Gamma_n u_n$$

where Γ_n and u_n are selected empirically to account for the cumulative random errors caused by all random excitation during the intervals between the observations.

Parameter estimation

In various applications there may be a number of parameters to be estimated in addition to the dynamic state variables. Suppose the parameters are designated as b_i $(i = 1, \ldots, m)$ and arranged in a vector

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

Then the discrete-time dynamics can be represented as

$$x_{n+1} = f(x_n, b, u_n) \qquad (3.21)$$

Likewise the observations can be expressed as

$$y_n = h(x_n, b, v_n) \qquad (3.22)$$

These equations are general enough to include as special cases observation bias and constant (unknown) forcing terms. Each observation bias is represented by a parameter b_i which appears only in the observation equation. Each constant forcing term appears only in the dynamic equations unless it contributes to a direct measurement.

There is a standard trick for handling parameter estimation, namely to treat b as a part of an enlarged state vector of $k+m$ components.

$$z_n = \left[\frac{x_n}{b_n}\right]$$

Since b is a constant it evolves in accordance with the equation

$$b_{n+1} = b_n$$

The initial condition $\hat{b}_0$ for the estimate is taken as the expected value before any observations, usually zero. The standard extended linear algorithm (or any other nonlinear extension) is applied to estimate z_n; the resulting estimate contains the estimate $\hat{x}_n$ in the presence of (initially) unknown parameters are well as an estimate $\hat{b}_n$ of the unknown parameters. The method works effectively and, in fact, often is the key to the success of the Kalman filtering algorithm in practical applications which abound in uncertain parameters. In the sense that the filter by estimating the uncertain parameters adapts to their presence, the technique may be and, in fact, has been termed "adaptive."

One of the problems with this method of parameter estimation is that the dimension of the big vector z_n can be quite large and, as a result the calculations, may require the manipulation of large matrices. In addition to requiring considerable storage, the usual numerical difficulties associated with large-matrix calculations may be present. In an attempt to avoid such numerical difficulties, Friedland[17] developed a computation technique, which is mathematically equivalent to dealing with the large vector z_n and the associated covariance matrices, but in which the estimation of the parameter vector b_n is decoupled from the estimation of the state vector. In particular, the algorithm consists of computing the estimate $\bar{x}_n$ of the state as if the parameter vector b were equal to its expected value prior to any observations, say zero, and the corresponding residual vectors $\bar{r}_n = y_n - h(\bar{x}_n)$. These residuals are used as the input to a "parameter estimator" which produces the optimum estimate $\hat{b}_n$ of the parameter b. The estimate $\hat{b}_n$ is then multiplied by a matrix V_n to yield the corrected estimate:

$$\hat{x}_n = \bar{x}_n + V_n\hat{b}_n \tag{3.23}$$

In applying this method, there are usually parameters which cannot be estimated, i.e., $\hat{b}_n$ changes negligibly from its value prior to any observations. It is clear from (3.23) that such parameters need not be included in the state vector z_n or in the parameter vector b_n. It is cautioned, however, that the covariance matrix, say, $\bar{P}_n$ which accompanies the computation of $\bar{x}_n$ does *not* realistically represent the accuracy of the estimated state. When uncertain parameters are present, the actual covariance matrix of the state is given by

$$\hat{P}_n = \bar{P}_n + V_nM_nV_n' \tag{3.24}$$

where V_n is the matrix in (3.23) and M_n is the covariance matrix of the parameter vector, i.e.,

$$M_n = E[\hat{b}_n\hat{b}_n'] \tag{3.25}$$

When the uncertainty is large it is quite possible that $V_nM_nV_n'$ is of the same order of magnitude as $\bar{P}_n$ or perhaps even larger.

THE "DIVERGENCE" PROBLEM

Almost as soon as Kalman's recursive filtering algorithm was announced there were a number of investigators ready to apply it to their specific problem. It was quickly discovered that a practical difficulty of the method is that the estimate of the state may tend to diverge from the true state and, in fact, may tend to be ridiculous. Kalman showed that the filtering algorithm is stable, i.e., erroneous covariance matrices for the dynamic noise and the observation noise will not cause the estimate to diverge from the true state. Consequently, the divergence could not be attributed to an inherent instability of the algorithm. Other explanations for the divergence problem had to be found. It became evident that the divergence problem cannot be attributed to a single source. Sometimes it may be due to difficulty in propagating the estimate of the state, in accordance with (3.10) and (3.11); sometimes in propagating the covariance matrix in accordance with (1.6) and (1.7); sometimes both.

Modeling errors

Errors in propagating the estimate of the state may result because of numerical errors in integrating the dynamic equations $\dot{x} = f(x)$, or because the dynamic equations do not really represent the true behavior of the process. The state of the art in numerical integration having been very advanced by the time the recursive filtering algorithm was introduced, the effect of numerical errors could reasonably be discarded. The possibility that the differential equations being integrated, however, do not accurately represent the true physical behavior of the process, however, is not so easily dismissed. The true behavior of the process may not be sufficiently well understood to be modeled accurately, or else an accurate dynamic model may be impractically complicated. The consequence of these "modeling errors" could very well be the divergence of the estimated state from the true state. Possible cures for the modeling errors which have been applied with success in various applications include:

- Representation of the modeling errors by "pseudo-noise"
- Bounding of the covariance matrix
- Use of finite—or fading-memory filtering
- Selective improvement of modeling.

The performance of the Kalman filter is achieved through the correlation of the state of the process at a given time to the state at earlier times. This correlation is established through the dynamics of the process. It follows that if the dynamic model is incorrect, the correlations assumed to be developed may be unreliable, and their use could be dangerous. Any noise present in the process excitation retards the development of this correlation, and it is thus reasonable to represent uncertainty in the model as having the same effect as would be produced by noise. The covariance matrix of this "pseudo-noise" is selected so that the standard deviation in the latter is about equal to the level of uncertainty in the dynamic model.

A technique related to the use of pseudo-noise is *a priori* covariance matrix bounding. The rationale for this technique is that, on physical grounds, we would regard it as impossible to achieve better than a specified accuracy in estimating some of the state variables. If the covariance matrix indicates that higher accuracy is being achieved, the matrix is increased to the *a priori* lower limit. Thus, instead of adding the covariance matrix $\Gamma_nQ_n\Gamma_n'$ corresponding to a known dynamic noise level at each step, this matrix is adjusted so that $\tilde{P}_{n+1} = \Phi_n\hat{P}_n\Phi_n' + \Gamma_nQ_n\Gamma_n'$ does not decrease below a specified lower limit. Similarly, the observation-updated covariance matrix $\hat{P}_n = \tilde{P}_n - \tilde{P}_nH_n'(H_n\tilde{P}_nH_n' + R_n)^{-1}H_n\tilde{P}_n$ can be kept from falling below a specified limit.

Another point of view on modeling uncertainties is represented by the idea that the validity of the dynamic model may be of limited duration. For example, there may be certain long-term trends which the model does not accurately treat, but near-term behavior may be modeled quite accurately. If this is the case, it would follow that data that are older than the interval over which the model is valid ought be discarded. In other words, the "memory" of the filter should be limited to the interval of validity of the model. The Kalman filtering algorithm, however, has infinite memory, in the sense that that the estimate at any instant of time depends on every observation datum ever processed. To be sure, the importance of each datum to the total estimate diminishes as more data are processed, but there is no way of entirely eliminating dependence on data older than a given time. Moreover, in a process with unstable dynamics, the importance of early data may be emphasized disproportionately. To avoid the dependence on all past data, the use of "finite memory" filters has been proposed by various investigators. The general idea is that the estimate $\hat{x}_n$ of the state x_n would be based on a finite amount of past data. For example, the optimum estimate could be defined by

$$\hat{x}_n = [x_n \mid y_n, y_{n-1}, \ldots, y_{n-N+1}] \tag{4.1}$$

i.e., the conditional mean given the most recent N observations: each time a new observation is made, the oldest observation is deleted. One method of obtaining a finite-memory estimate is to use a non-recursive algorithm. For example, assuming no dynamic noise, and following the development in the second section, the least-squares estimate of x_n given $y_n, \ldots y_{n-N+1}$, for a linear process, is given by

$$\hat{x}_n = \Phi_{n-1}\Phi_{n-2} \ldots \Phi_{n-N+1} x_{n-N+1}^{(n)} = F_{n,N} x_{n-N+1}^{(n)}$$

where

$$x_{n-N+1}^{(n)} = (M_{n,N}' W_{n,N} M_{n,N})^{-1} M_n' W_{n,N} Y_{n,N} \tag{4.2}$$

and

$$M_{n,N} = \begin{bmatrix} H_n\Phi_{n-1} & \cdots & \Phi_{n-N+1} \\ H_{n-1}\Phi_{n-2} & \cdots & \Phi_{n-N+1} \\ & \vdots & \\ H_{n-N+2} & \Phi_{n-N+1} & \\ H_{n-N+1} & & \end{bmatrix} \quad Y_{n,N} = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-N+1} \end{bmatrix}$$

Representation of (4.2) in recursive form appears to be difficult. Jazwinski[6] has a discussion of the implementation of algorithms for processing a finite amount of past data.

Instead of dropping all dependence on data older than a given number of samples, various investigators have suggested that a gradual diminution of importance of the data be assumed as the age of the data is increased. This can be accomplished by assuming that the noise covariance matrix increases from its "nominal" value in proportion to the number of samples elapsed. The computational advantage of the assumption of gradual diminution of data importance is that the data processing can usually be represented by a recursive algorithm. For example, Tarn and Zaborszky,[18] have shown that if the effective covariance matrix is taken as $s^l R_n (s > 1)$ where l is the number of elapsed samples the implementation is achieved by simply multiplying the term $\Phi_{n-1}\hat{P}_{n-1}\Phi_{n-1}'$ in (1.6) by s. Improvement by as much as a factor of 3 has been demonstrated in simulation[18] using a scale factor s in the vicinity of 1.1. The particular simplicity of this "cure" would appear to make it worth considering in instances of suspected divergence.

Perhaps the best treatment of modeling errors is to use a superior model. As already noted, this may not be always possible either because a better model may not be available, or because a better model, although available, may be too complicated. Very little can be done in the former case; in the latter case, however, it might be possible to use the recursive filtering method to reduce the complexity of the model to achieve a suitable balance between performance and complexity. The most complete model is formulated, and, in a series of simulation runs, the model is simplified until further simplifications are found to result in significant degradation in performance.

Covariance matrix computational errors

In a linear process the covariance matrices $\tilde{P}_n$ and $\hat{P}_n$ are independent of the estimates $\tilde{x}_n$ and $\hat{x}_n$, and can thus be computed in advance and stored in the computer, although storage problems may make if preferable to compute these "on-line" as the filter operates. In the extended Kalman filter, however, the covariance matrices depend on the actual estimates, and the real-time computation is unavoidable. The real-time computation of the covariance matrix may result in numerical errors which, if not corrected, could cause unsatisfactory filter performance. The basic difficulty is the tendency of the covariance matrix to lose its positive-definite character. It is emphasized that this is strictly a *numerical* problem, unrelated to any modeling errors which may (or may not) also be present. In other words, even if the transition matrix Φ_n and the observation sensitivity matrix H_n are not correct, errors in these matrices in themselves cannot cause the covariance matrix to become indefinite. This fact is easy to prove. Suppose $\tilde{P}_n$ is positive-definite. Then its inverse $\tilde{P}_n^{-1}$ exists, and is positive-definite. Consequently,

$$\hat{P}_n^{-1} = \tilde{P}_n^{-1} + H_n' R_n^{-1} H_n \tag{4.3}$$

is positive-definite since $H_n' R_n^{-1} H_n$ is (at least) positive semi-definite, and the sum of a positive definite matrix and a positive semi-definite matrix is positive definite. Thus $\hat{P}_n = (\hat{P}_n^{-1})^{-1}$ exists and is positive definite. Now the transition matrix of a dynamic process is nonsingular, so $\Phi_n \hat{P}_n \Phi_n'$ is also positive definite and hence

$$\tilde{P}_{n+1} = \Phi_n \hat{P}_n \Phi_n' + \Gamma_n Q_n \Gamma_n'$$

is positive definite for any positive semi-definite Q_n.

This reasoning makes it clear that if $\tilde{P}_0$ is positive definite then all subsequent values $\tilde{P}_n$ and $\hat{P}_n$ $(n=0, 1, 2, \ldots)$ are *theoretically* positive-definite. Theoretical positive-definiteness, unfortunately does not assure freedom from numerical difficulties. In particular, the matrix $\hat{P}_n^{-1}$ and its inverse $\hat{P}_n$ are very frequently ill-conditioned matrices, in the sense that they may have numerically large elements but a relatively small determinant. In particular, suppose that at some step $\tilde{P}_n^{-1}$ is very small (i.e., $\tilde{P}_n$ is very large) but that $H_n' R_n^{-1} H_n$ is large, i.e., R_n is small—this corresponds to the case in which the sensors are very accurate. Suppose also that not all the state variables are measured; then $H_n' R_n^{-1} H_n$ will be a singular (positive-semidefinite) matrix and the sum

$$\hat{P}_n^{-1} = \tilde{P}_n^{-1} + H_n' R_n^{-1} H_n$$

will be ill-conditioned. The computation of its inverse may be expected to be fraught with error.

One obvious possible solution to the problem of ill-conditioned covariance matrices is to increase the precision of computation. When double-precision hardware is available, the only significant cost is in storage, and thus double-precision calculations may be a feasible cure. In instances when double-precision hardware is not available, or when numerical difficulties traceable to ill-conditioned matrices persist even when double-precision arithmetic is used, it is necessary to seek alternative methods. Various *ad hoc* "fixes" have been used with success. One method is to add fixed small positive numbers to the diagonal elements of the covariance matrices as they are computed, on the theory that if a matrix is nearly singular, the addition of a small positive diagonal matrix makes it less so. Another method, entailing more calculation, is to test the covariance matrix for positive-definiteness, and, if at any stage, the matrix becomes indefinite, to add a small positive definite matrix just large enough to make the result slightly positive-definite. Since the covariance matrix is theoretically symmetrical, some users employ a "symmetrization" operation in which P_n is replaced by $(P_n + P_n')/2$, which is, of course, a symmetric matrix. The advantage of symmetrization is not certain, however, because the real parts of the eigenvalues of a matrix are unchanged by this operation, and, as a result, indefiniteness is not affected by symmetrization. On the other hand, numerical difficulties which might be obvious upon inspection of the unsymmetrized covariance matrix could be masked by the symmetrization operation.

A systematic method of possibly overcoming the ill-conditioning problem is to use the inverse of the covariance matrix (which some investigators have called the "information matrix"). In particular, let

$$\tilde{\mathfrak{b}}_n = \tilde{P}_n^{-1}, \qquad \hat{\mathfrak{b}}_n = \hat{P}_n^{-1}$$

Then, using (4.3)

$$\hat{\mathfrak{b}}_n = \tilde{\mathfrak{b}}_n + H_n R_n^{-1} H_n' \tag{4.4}$$

and using (2.9), which implies $K_n = \hat{P}_n H_n' R_n^{-1}$, we obtain

$$K_n = \hat{\mathfrak{b}}_n^{-1} H_n' R_n^{-1} \tag{4.5}$$

The inverse R_n^{-1} of the observation noise covariance matrix can be given as input data instead of R_n. Thus the information matrix $\hat{\mathfrak{b}}_n$ is obviously no worse conditioned than $\tilde{\mathfrak{b}}_n$ and its inversion to obtain the gain matrix K_n from (4.5) should cause no difficulty. Moreover, any error which arises in the computation of K_n is not propagated to later steps in the filter operation. In the absence of dynamic excitation noise $(Q_n \equiv 0)$, the time update of the information matrix is given by

$$\tilde{\mathfrak{b}}_{n+1} = \tilde{P}_{n+1}^{-1} = \Psi_n \hat{\mathfrak{b}}_n \Psi_n' \tag{4.6}$$

where

$$\Psi_n = \Phi_n^{-1\prime}$$

Because Ψ_n' is the inverse of a *transition matrix*, it can be computed by numerical integration of the *adjoint* differential equation

$$\dot{\Psi} = -\left[\frac{\partial \alpha}{\partial x}\right]' \Psi$$

and entails no matrix inversion. Thus, when there is no process excitation present, the computational requirements of this method are comparable to the method of covariance matrices; the major difference is that now it is necessary to invert an $n \times n$ matrix $\hat{\mathfrak{b}}_n$ to obtain the gain matrix instead of an $r \times r$ matrix $(H_n \tilde{P}_n H_n' + R_n)$. Since r, the dimension of the observation vector is usually much smaller than n, the additional computer time needed to realize the calculations using (4.4) through (4.6) may be rather considerable.

When random excitation is present in the dynamics, i.e., $\Gamma_n \neq 0$ in (1.1), then (4.6) is not valid; instead, the correct relationship is

$$\tilde{b}_{n+1} = \Psi_n[\hat{b}_n^{-1} + \Psi_n'\Gamma_n Q_n \Gamma_n' \Psi_n]^{-1}\Psi_n' \qquad (4.7)$$

and entails an additional matrix inversion. In most cases, however, the additive noise as represented by $\Gamma_n Q_n \Gamma_n'$ would be expected to be quite small, and the inverse of the bracketed matrix in (4.7) can be approximated by $\hat{b}_n - \hat{b}_n \Psi_n' \Gamma_n Q_n \Gamma_n' \Psi_n \hat{b}_n$. Thus (4.7) would be approximated by

$$\tilde{b}_{n+1} \doteq \Psi \hat{b}_n \Psi_n' - \Psi_n \hat{b}_n \Psi_n' \Gamma_n Q_n \Gamma_n \Psi_n \hat{b}_n \Psi_n' \qquad (4.8)$$

The computation of the additional term in (4.8) not present in (4.7) entails two additional matrix multiplications, which may not prove significantly better to evaluation of (4.7).

It is worth pointing out that the matrices to be inverted in all aspects of Kalman filtering are symmetric and (theoretically) positive-definite. Use of inversion algorithms that are specific for such matrices might be desirable. In particular, the Gauss-Seidel iterative method for inverting $\hat{b}_n^{-1} + \Psi_n' \Gamma_n Q_n \Gamma_n' \Psi_n$, with $\hat{b}_n$ used to start the iteration, would very likely converge in one or two iterations.

Another method for dealing with the ill-conditioned matrices that arise in the recursive least-squares algorithm, has been termed the "square-root method." The method is based on representing each covariance matrix as the product of a lower triangular matrix and its transpose, i.e.,

$$P_n = S_n S_n', \qquad \hat{P}_n = \hat{S}_n \hat{S}_n' \qquad (4.9)$$

where S is a matrix in which $S_{ij} = 0$ for $j > i$. When P and S are scalars $S = \sqrt{P}$, hence the name of the method. (In a strict sense a lower triangular matrix S satisfying (4.9) is *not* the square root of P. A true matrix square root, say W, satisfies $W^2 = P$, and when P is symmetric, W is also symmetric and thus cannot be lower triangular.)

A survey of the background and current status of the square-root method is contained in a recent paper by Kaminski, Bryson and Schmidt.[19] In the absence of process noise, the square root algorithm may be expressed by the formulas

$$K_n = \tilde{S}_n F_n G_n'^{-1} G_n^{-1} \qquad (4.10)$$

where

$$F_n = \tilde{S}_n' H_n' \qquad (4.11)$$

and

$$G_n G_n' = R_n + F_n' F_n \qquad (4.12)$$

i.e., G_n is the (unique) lower triangular factor of $R_n + F_n F_n'$ and G_n' is the corresponding upper triangular factor. The updating of the "square root" matrix is done using

$$\hat{S}_n = \tilde{S}_n - \tilde{S}_n F_n G_n^{-1\prime}(G_n' + V_n)^{-1} F_n' \qquad (4.13)$$

$$\tilde{S}_{n+1} = \Phi_n \hat{S}_n$$

where

$$V_n V_n' = R_n$$

Only triangular matrices are inverted in (4.10) and (4.13) which is a considerable advantage. The major computational chore is in the factorization of the matrix $R_n + F_n F_n'$. This factorization problem is somewhat analogous to the spectral factorization problem which arises in Wiener filtering (see Friedland[20] for a discussion of this connection as well as for independent development of the Cholesky-Banachiewicz algorithm[19] for accomplishing the factorization).

When process noise is present, a more complicated algorithm is needed, but the basic ideas are the same.

The large consumer of computer capacity is the triangularization operation implicit in (4.12). The efficiency of the square-root method thus depends on the algorithm used to find G_n. The authors of Reference 19 have made an extensive study of suitable algorithms and conclude that Householder's algorithm is adaptable to this problem. A count of operations made in Reference 19 shows that the square-root algorithm may typically require only as much as 30 percent of additional computer time over that required by the conventional (Kalman-Bucy) algorithm and even less time than use of the information-matrix algorithm. The numerical advantage of the square-root method is claimed to be equivalent to doubling the precision of calculations: ". . . a single precision square-root filter would provide the accuracy equivalent of a double precision conventional filter in ill-conditioned problems."[19]

CONCLUDING REMARKS

No survey of the first decade of Kalman filtering would be complete without a discussion of applications which were investigated almost as soon as the theory appeared. (Since the Swerling paper,[5] which predates Kalman's paper by a year, deals with an application, it would be reasonable to assert that applications were investigated even before Kalman's theory appeared.)

Coming at the time of the rapid expansion of the country's aerospace program, it is understandable that the majority of applications were in the aerospace field. G. Smith's very timely paper[13] aroused considerable interest when it was presented and perhaps was to a very large measure responsible for the rapid assimilation of the method by aerospace engineers. The application of the method to orbit and trajectory determination described in the book[21] of a notable expert, R. Battin, helped to establish the importance of the theory. Since the earliest papers, which demonstrated feasibility of the method largely through simulations, the number of applications which have been studied is overwhelming. Hundreds of papers demonstrating potential performance have been presented at technical meetings and have been published in the journals of technical societies. Moreover, the published studies represent only a small fraction of those actually performed. Many others are documented in company internal technical reports or in final reports of limited circulation (many also classified) prepared under government contract. The recursive filtering algorithm is so simple that almost anyone can develop a simulation program. Many organizations evidently have done so.

Application of the Kalman filtering technique to processing of real-world data has naturally been more limited than application studies conducted through simulation. Nevertheless, the number and importance of real-world applications of this technique is most impressive. Those who have listened to the broadcasts of Apollo flight conversations must have heard reference to "state-vector updates"—clearly the language of Kalman filtering. And also the substance. A review of the extensive use of Kalman filtering in the Apollo program by Battin and Levine is contained in a compendium[22] of theory and applications papers that were presented at a 1970 AGARD meeting. Another important application of Kalman filtering has been to aided-inertial navigation. The Kalman filter, in this application, is used to optimally mix data from an inertial platform which has very low inherent short-term errors, but which can build up substantial secular drifts, and non-inertial (e.g., radio ranging, LORAN, doppler) navigation aids which have opposite characteristics, namely that they are relatively drift-free but typically quite noisy. The Kalman filtering algorithm is implemented using an airborne digital computer on the C5-A military transport. A description of the system is given in Reference 22. Inertial navigation applications have advanced to the point that implementation of the Kalman filtering algorithm is now a significant design consideration, in particular, with regard to the design of airborne digital computers to be used in such systems.

Kalman filtering applications appears to be significantly less widespread outside of the aerospace field. Perhaps one reason for this is that dynamic models which play a large role in Kalman filtering are not well established for processes if interest, especially in very large-scale systems (e.g., urban dynamics, ecological systems, etc.). The existence of an analytically-defined dynamic model would appear to be a minimum prerequisite for application of the recursive filtering algorithm, unless methods can be found for using the algorithms in the absence of such models. In the field of industrial processes, Kalman filtering applications have been reported in conjunction with direct digital control. An early, and by now rather celebrated, application in the Swedish paper-making industry was presented by Åström.[23] Other applications in the paper industry, the cement industry, the metallurgical industry, the electric power industry, have been reported or proposed.

The recursive filtering technique developed by Kalman and Bucy would appear to be mature and vigorous despite its youth. The easy theoretical problems seem to have been solved, and the reasons why the difficult problems are difficult have been discovered; approximate methods have been developed to facilitate engineering applications and to develop insights; applications in diverse fields have been studied and often implemented; the basic results are teachable and are included in many curricula in systems engineering. Like many engineering analysis techniques of the past, some of the glamor of Kalman filtering is likely to wear off. What remains, however, will continue to be useful for many years to come.

REFERENCES

1 R E KALMAN
A new approach to linear filtering and prediction problems
Trans ASME J Basic Eng Series D Vol 82 pp 34-45 March 1960

2 R E KALMAN
New methods in Wiener filtering theory
In Proc 1st Symp Engineering Applications of Random Function Theory and Probability J L Bogdanoff and F Kozin Eds New York Wiley 1963 pp 270-388

3 R E KALMAN R BUCY
New results in linear filtering and prediction theory
Trans ASME J Basic Eng Series D Vol 83 pp 95-108 March 1961

4 N WIENER
The extrapolation, interpolation, and smoothing of stationary time series with engineering applications
New York Wiley 1949

5 P SWERLING
First order error propagation in a stagewise smoothing procedure for satellite observations
J Astronaut Sci Vol 6 pp 46-52 Autumn 1959

6 A H JAZWINSKI
Stochastic processes and filtering theory
New York Academic Press 1970

7 R L STRATONOVICH
Conditional Markov processes
Theory of Probability and Applications Vol 5 pp 156-178 1960

8 H J KUSHNER
On the dynamical equations of conditional probability density functions with applications to optimum stochastic control theory
J Math Anal Appl Vol 8 p 332 1964

9 H J KUSHNER
On the differential equations satisfied by conditional probability densities of Markov processes
SIAM J on Control Vol 2 pp 106-119 1964

10 R S BUCY
Nonlinear filtering theory
IEEE Trans on Automat Contr Vol 10 p 198 April 1965

11 T KAILATH
An innovations approach to least-squares estimation—Part I: Linear filtering in additive white noise
IEEE Trans Automat Contr Vol AC-13 pp 646-655 December 1968

12 T KAILATH P FROST
An innovations approach to least-squares estimation—Part II: Linear smoothing in additive white noise
IEEE Trans Automat Contr Vol AC-13 pp 655-660 December 1968

13 G SMITH
Multivariable linear filter theory applied to space vehicle guidance
Presented at SIAM Symposium on Multivariable System Theory Cambridge Mass November 1962

14 H COX
On the estimation of state variables and parameters for noisy dynamic systems

15 B FRIEDLAND I BERNSTEIN
Estimation of the state of a nonlinear process in the presence of nongaussian noise and disturbances
J Franklin Institute Vol 281 pp 455-480 July 1966

16 R S BUCY K D SENNE
Digital synthesis of nonlinear filters
Automatica Vol 7 No 3 pp 287-298 1971

17 B FRIEDLAND
Treatment of bias in recursive filtering
IEEE Trans Automat Contr Vol AC-14 pp 359-367 August 1969

18 T J TARN J ZABORSZKY
A practical, nondiverging filter
AIAA J Vol 8 pp 1127-1133 June 1970

19 P G KAMINSKI A E BRYSON JR S F SCHMIDT
Discrete-square root filtering: A survey of current techniques
IEEE Trans on Automat Contr Vol AC-10 No 6 pp 727-735 December 1971

20 B FRIEDLAND
Least squares filtering and prediction of nonstationary sampled data
Inform Contr Vol 1 pp 297-313 1958

21 R H BATTIN
Astronautical guidance
New York: McGraw-Hill 1964

22 C T LEONDES ed
Theory and applications of Kalman filtering
AGARDograph No 139 NATO Advisory Group for Aerospace Research and Development February 1970

23 K J ÅSTRÖM
Computer control of a paper machine—An application of linear stochastic control theory
IBM J Res Develop Vol 11 pp 389-405 1967

On computational methods for dynamic and static optimization*

by D. H. JACOBSON

Harvard University
Cambridge, Massachusetts

INTRODUCTION

In the 1960s the joint impact of fast digital computers and the space age stimulated the ingenuity of a number of researchers in optimization and prompted them to invent new and/or better (faster, more elegant) methods for solving, numerically, optimization problems of different types. Two areas that received much attention were trajectory optimization and parameter optimization. That is, the computational problems of determining optimal trajectories (generally, functions of time) and optimal parameter values by minimizing (maximizing) appropriate performance criteria.

Many of the (very successful) algorithms developed were heuristic in that they were good ideas translated skillfully, albeit non-rigorously in the mathematical sense, into algorithms and computer programs. More recently, more effort has been assigned to the theoretical problems of convergence and rate of convergence of algorithms. But, and this is an important point, new powerful algorithms rarely arise out of purely abstract treatments while on the other hand good heuristic methods seldom manifest themselves unless their inventors have at least some deep, albeit informal, understanding of theories of convergence which would enable them to propose an algorithm which is "unlikely not to converge." Thus, good heuristics and theory of algorithms should complement one another in the invention and development of numerical methods.

Certain trajectory optimization algorithms that manifested themselves, as powerful numerical methods for solving real aerospace problems are described in References 1 and 5. The reader should acquaint himself with these because, though some of the methods are dated, insight into the development of computational methods in optimization can be gained and, moreover, certain of the second-variational algorithms remain as attractive methods.

Our work in trajectory optimization (or, more generally, computation of optimal controls) has been concerned with the development of a class of algorithms and a methodology of computation based on Dynamic Programming. The approach which is called Differential Dynamic Programming (DDP) arose as a consequence of an early stimulating paper[6] and is quite extensively described in Reference 7. While no theorems of convergence are given in Reference 7, conditions and proofs for continued descent are provided, and the algorithms have successfully solved a number of substantial numerical problems. Thus, there is still room for theoretical extensions in the form of convergence proofs and rate of convergence studies, though these will be nontrivial. Mayne's[8] recent approach to DDP may prove to be useful in these theoretical problems.

In parameter optimization which, basically, involves choosing the values of n variables to minimize a scalar function, the most famous and widely used methods are given in References 9 and 10. These methods are based on quadratic modelling of the function to be minimized and hence are quasi-Newton in type. Interestingly Fletcher and Powell's algorithm appeared in 1963, and has been very popular, but it was not until 1971 that a proof of convergence was offered by Powell.[11]

Our interest in the function minimization problem led us to propose a new class of algorithms which is based upon the concept of an homogeneous function.[12] Our algorithm[12] is but one possible example of the use of the homogeneous model and it is likely that others will be forthcoming. The algorithm is roughly at the stage that Fletcher and Powell's was circa 1964 in that computational experience suggests that it is attractive, while convergence and rate of convergence studies are yet to come.

A recent book which is appealing in its novelty of approach to the theoretical questions of convergence

* This research is supported by the Joint Services Electronics Program on Contract N00014-67-A-0298-0006.

and rate of convergence is Polak's.[13] Here, general theorems on convergence are presented and applied to known and new algorithms. Though this general theorem approach is appealing and useful it is not a panacea; indeed, some effort appears to be needed to twist the algorithm described in Reference 12 into the framework and verification of a number of required conditions seems to be nontrivial.[14]

With this background and philosophy noted, we continue with subsequent sections of this paper which describe briefly the notions of DDP for control optimization, and homogeneous modelling for function optimization. These are two approaches which seem to be attractive and which, as outlined above, offer potential for future research.

OPTIMAL CONTROL COMPUTATION

Problem formulation

We are concerned with the problem of finding a control function $u(\cdot)$ which minimizes

$$V(x_0, u(\cdot), t_0) = \int_{t_0}^{t_f} L(x, u, t)\, dt + F[x(t_f)] \quad (1)$$

subject to the dynamic system constraints

$$\dot{x} = f(x, u, t); \quad x(t_0) = x_0 \quad (2)$$

and the terminal and control constraints

$$\psi(x(t_f)) = 0 \quad (3)$$

$$g(u(t)) \leq 0 \quad (4)$$

Here, $L: R^n \times R^m \times R^1 \rightarrow R^1$, $F: R^n \rightarrow R^1$, $f: R^n \times R^m \times R^1 \rightarrow R^n$, $\psi: R^n \rightarrow R^s$, $g: R^m \rightarrow R^q$, and all these functions are assumed to be three times continuously differentiable in each argument. The initial and final times t_0, t_f are assumed to be given.

In general, one is unable to find the absolute minimum of V (some sort of global search technique would be required) unless V is a convex functional of $u(\cdot)$ which would exclude the possibility of there existing a number of local minima having different V values. Typically what one can find is a stationary point of V or, at best, a local minimum of V.

It is known that a necessary condition (Pontryagin's Principle) for $V(x_0, u(\cdot), t_0)$ to have a relative minimum with respect to $u(\cdot)$ at $u(\cdot) = u^0(\cdot)$ is that there exist multipliers $\rho_0 \in R_+^1$, $\nu \in R^s$, $\lambda(\cdot)$ absolutely continuous, not all zero, such that

$$-\dot{\lambda} = H_x; \quad \lambda(t_f) = \rho_0 F_x[x(t_f)] + \nu^T \psi_x[x(t_f)] \quad (5)$$

where

$$H(x, u, \lambda, t) = \rho_0 L(x, u, t) + \lambda^T f(x, u, t) \quad (6)$$

and

$$u^0(t) = \arg \min_{u(t) \epsilon U} H(x, u, \lambda, t) \quad (7)$$

where

$$U \equiv \{u(t) : g(u(t)) \leq 0\} \quad (8)$$

Usually one assumes that the problem is normal (i.e., that the conditions can be satisfied with $\rho_0 \neq 0$) so that ρ_0 can be set equal to unity.

Thus, at the very least, one is interested in determining a control function which satisfies the above necessary condition.

Certain algorithms [2], [3] are based directly on second-order expansions of the above optimality conditions, but we prefer to use the notion of DDP which is intuitively more appealing and gives rise to algorithms which are significantly different from those arising from expansions of conditions (5)-(7). Of course, with hindsight, it is possible to derive these algorithms by expanding (5)-(7) but it is the Dynamic Programming formulation which gave rise to these in the first place (see Reference 15 for some discussion of this).

For the purpose of this presentation we shall assume $g(\cdot) = 0$ (i.e., no control constraints) and at the outset we shall assume $\psi(\cdot) = 0$ (no terminal constraints) though we shall allow $\psi(\cdot) \neq 0$ later in the paper (see Reference 7 for DDP algorithms that handle control constraints).

We define the optimal value function $V^0(x, t)$ by

$$V^0(x, t) \overset{\Delta}{=} \min_{u(\tau), t \leq \tau \leq t_f} \int_t^{t_f} L(x, u, t)\, dt + F(x(t_f)) \quad (9)$$

On assuming that $V^0(x, t)$ is twice continuously differentiable with respect to x, t one can obtain the following (Bellman) partial differential equation for $V^0(x, t)$:

$$-(\partial V^0/\partial t)(x, t) = \min_u [L(x, u, t) + \{V_x^0(x, t)\}^T f(x, u, t)] \quad (10)$$

$$\overset{\Delta}{=} \min_u H(x, u, V_x^0, t) \quad (11)$$

where

$$V^0[x(t_f), t_f] = F[x(t_f)] \quad (12)$$

The solution of this partial differential equation is straightforward in the case of quadratic L, F and linear f (V^0 turns out to be a quadratic function of x) but, in general, it can be solved only numerically. As

$V^0: R^n \times R^1 \to R^1$, storage and computer time requirements become unmanageable for $n \geq 3$. This is in contrast to Differential Dynamic Programming which allows us to compute the optimal control $u^0(\cdot)$ for quite large n but does not yield the global solution of Bellman's equation (in many cases knowledge of this solution would be of doubtful value; often, all one wants is $u^0(t)$, $t \in [t_0, t_f]$).

Differential dynamic programming

Here we shall indicate the approach of DDP and refer the reader to Reference 7 for details of the algorithm. Basically, one guesses a nominal control function $\bar{u}(\cdot)$ which yields a nominal cost $V[x_0, \bar{u}(\cdot), t_0]$ and a nominal state variable trajectory $\bar{x}(\cdot)$. Estimates of $V_x^0[\bar{x}(t), t]$, $V_{xx}^0[\bar{x}(t), t]$ are obtained and this "local information concerning V^0" is used to yield an improved control function $u_1(\cdot) = \bar{u}(\cdot) + \delta u(\cdot)$ where $\delta u(\cdot)$ is such that

$$V(x_0, u_1(\cdot), t_0) < V(x_0, \bar{u}(\cdot), t_0) \tag{13}$$

The DDP algorithm is then used iteratively until a stopping criterion is satisfied; e.g.,

$$\int_{t_0}^{t_f} H_u{}^T H_u \, dt \leq \epsilon \tag{14}$$

Fixed end-point problem

Here we allow terminal constraints

$$\psi[x(t_f)] = 0 \tag{15}$$

In order to solve this problem we make the assumption that

$$V[x_0, u(\cdot), b, t_0] \overset{\Delta}{=} V[x_0, u(\cdot), t_0] + b^T \psi[x(t_f)] \tag{16}$$

has a minimum w.r.t. $u(\cdot)$ for all $b \in R^s$. Note that a well-known sufficient condition for this to be true is that $L(x, u, t)$ be strictly convex in $[x, u]^T$, F be convex in $x(t_f)$, f be linear in x, u, and ψ be linear in $x(t_f)$.

If the above assumption and certain other conditions [7] are satisfied then the optimal solution is given by

$$\max_{b \in R^s} V^0(x_0, b, t_0) \tag{17}$$

where

$$V^0(x_0, b, t_0) \overset{\Delta}{=} \min_{u(\cdot)} \int_{t_0}^{t_f} L(x, u, t) \, dt + F[x(t_f)] + b^T \psi[x(t_f)] \tag{18}$$

DDP extends easily to this situation by computing estimates of $V_b^0(x_0, \bar{b}, t_0)$, $V_{bb}^0(x_0, \bar{b}, t_0)$ to enable the nominal value of $\bar{b}$ to be improved iteratively.[7]

DDP algorithms for solving problems with control variable constraints are presented in Reference 7 but the important case of control problems with state variable inequality constraints has not been solved; i.e., the case where the trajectory must satisfy

$$S(x(t), t) \leq 0, \quad \forall t \in [t_0, t_f], \quad S: R^n \times R^1 \to R^1 \tag{19}$$

Certain optimality conditions for this problem [16], obtained recently, could stimulate activity in this important area of research.

FUNCTION MINIMIZATION

In the previous section we outlined the DDP approach for computing minima of functionals (dynamic optimization) and suggested a problem (state constrained) for future research. Here we wish to outline a new approach to the minimization of functions and to again suggest research directions.

Problem formulation

$$\underset{x}{\text{Minimize}}\, C(x), \quad C: R^n \to R^1 \tag{20}$$

Here C is assumed to be twice continuously differentiable and to have a unique stationary point, which is its minimum, at $x = x^*$ (in the case of functions having multi-minima and/or stationary points, available algorithms will generally find "the nearest one").

Available methods

Apart from the classical steepest descent and Newton methods, the two most widely used methods are probably the conjugate gradient method[9] and Fletcher and Powell's method.[10] These methods have the attractive feature of converging in n steps to the minimum if $C(x)$ is a quadratic function,

$$C(x) = \tfrac{1}{2}(x - x^*)^T Q (x - x^*) + \bar{\omega} \tag{21}$$

When applied to non-quadratics (computed examples show that) these methods are impressive in their speed of convergence provided that the non-quadratic looks quadratic in the neighborhood of the minimum [i.e., provided $C_{xx}(x^*) > 0$]. There is a host of other methods based on quadratic models (Quasi-Newton methods) and each has its advantages and disadvantages when compared with References 9 and 10; see, for example

Reference 17. The important point is that researchers in this field have exploited and continue to exploit, admittedly cleverly, the quadratic model. Thus, the homogeneous model, described below, seems to be a departure from conventional approaches (though a certain method that arises from the homogeneous model is closely related to Newton's method).

Homogeneous function as a model for minimization algorithms

The continuously differentiable function $C(x)$ is said to be homogeneous of degree γ if

$$C(x) = \gamma^{-1}(x-x^*)^T(\partial C/\partial x)(x) + \bar{\omega} \qquad (22)$$

Clearly a quadratic function is homogeneous of degree 2. An example of a non-quadratic homogeneous function is

$$C(x) = [\tfrac{1}{2}(x-x^*)^T Q(x-x^*)]^p, \quad p>1, Q>0 \qquad (23)$$

Since $C_{xx}(x^*)=0$ convergence of conjugate gradient methods is particularly poor for this type of function [12].

Multiplying (22) by γ yields

$$\gamma C(x) = (x-x^*)^T(\partial C/\partial x)(x) + \omega, \quad \gamma\bar{\omega} \overset{\Delta}{=} \omega \qquad (24)$$

Now, the attractiveness of this model is apparent; namely, the unknowns x^*, γ, ω appear *linearly* and can be solved for, given $n+2$ linearly independent vectors

$$[C(x_i), (\partial C/\partial x)(x_i), -1], \quad i=1, \ldots n+2 \qquad (25)$$

Our new algorithm, for general functions, described in Reference 12 is one possible procedure for iteratively updating the estimates of x^*, γ, ω as new data points become available. Numerical experience indicates that the algorithm is at worst competitive with Fletcher and Powell's and, in a number of representative examples, is almost twice as fast in terms of number of evaluations of the pair $[C(x), (\partial C/\partial x)(x)]$.

Though the class of homogeneous functions is larger than, and includes, quadratic functions, a suitable modification of Newton's method will find the minimum, x^*, in one step. Differentiating (24) with respect to x yields

$$[(\partial^2 C/\partial x^2)(x)](x-x^*) = (\gamma-1)(\partial C/\partial x)(x) \qquad (26)$$

whence, if the required inverse exists and is positive definite,

$$x^* = x - (\gamma-1)[(\partial^2 C/\partial x^2)(x)]^{-1}(\partial C/\partial x)(x) \qquad (27)$$

Note that this is a Newton step modified by the scale factor $(\gamma-1)$. Thus Newton's method, which minimizes a quadratic in one step, minimizes a homogeneous function if a step length of $(\gamma-1)$ is used. This appears to be unknown in the function minimization literature though analogous behavior, in the problem of locating a repeated root of an equation, was recognized in 1870 by Schröder. Unfortunately, methods which generate estimates of the second derivative matrix (inverse) from past data, for example Fletcher and Powell, do not preserve this valuable property; this is another motivation for estimating x^*, γ, ω directly from (24), as is done in Reference 12.

CONCLUSION

In this paper our intent is to place in perspective certain approaches to the development of optimization techniques, and to suggest certain research directions.

Two areas of recent activity, namely dynamic and static optimization via DDP and homogeneous modelling, are introduced. DDP and homogeneous modelling are two novel approaches to optimization which could be exploited further.

REFERENCES

1 H J KELLEY
Methods of gradients
In Optimation Techniques G Leitmann ed Academic Press New York 1962

2 H J KELLEY R E KOPP H G MOYER
A trajectory optimization technique based upon the theory of the second variation
In Progress in Astronautics and Aeronautics Vol 14 New York Academic Press 1964 pp 559-582

3 S R McREYNOLDS A E BRYSON JR
A successive sweep method for solving optimal programming problems
Proc 1965 Joint Automatic Control Conference No 6 p 551

4 H J KELLEY B R UZZELL S S McKAY
Rocket trajectory optimization by a second-order numerical technique
AIAA NASA-MSC Astrodynamics Conference Houston Texas December 12-14 1967

5 C W MERRIAM III
Optimization theory and the design of feedback controls
McGraw-Hill New York 1964

6 D Q MAYNE
A second-order gradient method of optimizing non-linear discrete time systems
Int J Control 3 1966 pp 85-95

7 D H JACOBSON D Q MAYNE
Differential dynamic programming
American Elsevier New York 1970

8 D Q MAYNE
Differential dynamic programming—A unified approach to the optimization of dynamic systems
To appear in Advances in Control Systems C T Leondes ed 1972

9 R FLETCHER C M REEVES
Function minimization by conjugate gradients
Computer Journal 7 July 1964 pp 149-154

10 R FLETCHER M J D POWELL
A rapidly convergent descent method for minimization
Computer Journal 6 1963 pp 163-168

11 M J D POWELL
On the convergence of a variable metric algorithm
J Inst Math Applic 7 1971 p 21

12 D H JACOBSON W OKSMAN
An algorithm that minimizes homogeneous functions of n variables in $(n + 2)$ iterations and rapidly minimizes general functions
J Math Anal Appl 1971 To appear

13 E POLAK
Computational methods in optimization—A unified approach
Academic Press New York 1971

14 E POLAK D H JACOBSON
Informal Talks Spring 1971

15 W F DENHAM
Review of Reference 7
IEEE Trans Automatic Control Volume 16 August 1971 pp 389-390

16 D H JACOBSON M M LELE J L SPEYER
New necessary conditions of optimality for control problems with state variable inequality constraints
J Math Anal Appl 35 1971 pp 255-284

17 B A MURTAGH R W H SARGENT
Computational experience with quadratically convergent minimization methods
Computer J Volume 13 May 1970 pp 185-194

Piecewise linear approximations of fewest line segments

by D. G. WILSON

Virginia Commonwealth University
Richmond, Virginia

INTRODUCTION

Schwerdtfeger[4,5] has considered the problem of interpolation and curve fitting over a given partition by a sum of continuous sectionally linear functions. He showed how to construct an orthogonal system of these functions over a given partition and how to use this system to obtain an approximation of a given function which is best in the least square sense.

Stone,[6] Bellman,[1] and Gluss[2] have analyzed various aspects of the problem of obtaining a piecewise linear approximation, not necessarily continuous, with a given number of line segments which is best in the least square sense.

More recently, Phillips[3] has given an algorithm for obtaining a continuous piecewise linear approximation to a function whose second derivative is of one sign. This algorithm gives an approximation which is of fewest segments among those with a given deviation. The output from this algorithm is a partition of the given interval and the parameters of a straight line over each subinterval of the partition.

We consider the problem of approximating a continuous sectionally linear function by another piecewise linear function, which may or may not be continuous, satisfying:

(a) the approximation is within a given deviation, which need not be constant, of the given function and
(b) the approximation is extremal in the sense that there is no other piecewise linear approximation which is within the same or lesser deviation and which has fewer line segments.

We give an algorithm which solves the problem as stated and a modification which gives a continuous piecewise linear, approximation which is within a given deviation of the given function and which is extremal among such approximations in the sense given above.

We thus solve the problem of fitting data known only at discrete abscissa values within a variable tolerance. Our algorithms can also be used to approximate continuous functions. However, in this case a partition of the interval of interest must be chosen first, the values of the function at these points computed, and an initial continuous piecewise linear approximation obtained by linear interpolation. It is evident that our algorithms cannot give a "good" fit if this initial approximation is not "good." It is also evident how to make this initial approximation "good." But it is not at all obvious how to make it how "good." We have not solved this problem in general. Perhaps a combination of Phillip's algorithm and ours could be used to solve the problem for continuous functions whose domains are separable into intervals over which the second derivative is of one sign.

In the second section we give a precise statement of the problem we have solved. Section three contains the basic algorithm and the modification. Section four contains the proofs which validate the algorithm. Section five contains some sample results. Section six acknowledges the assistance of others. A listing of a FORTRAN program which implements our algorithms is appended.

STATEMENT OF THE PROBLEM

Let P denote a partition of the finite interval $[a, b]$.

$$P = \{x_0, x_1, \ldots, x_n\},\ x_0 = a,\ x_n = b,\ i > j \Rightarrow x_i > x_j.$$

Let $f(x)$ be a continuous function defined on $[a, b]$ which is linear on each subinterval $[x_{i-1}, x_i] i = 1, \ldots, n$. Let $\delta(x)$ be the continuous sectionally linear extension to $[a, b]$, obtained by linear interpolation, of $\delta(x) = \delta_j$ when $x = x_j \in P$, where $\delta_j > 0$, $j = 0, \ldots, n$, are the permissible deviations at the points of P.

Then the problem is, given $u_0 = a$, determine m, p_i, q_i, and u_i, $i = 1, \ldots, m$ such that $F(x)$ given by:

$$F(x) = p_i x + q_i$$

when $x \in [u_{i-1}, u_i)$ for $i = 1, \ldots, m$ and

$$F(u_m) = p_m x + q_m,$$

satisfies $|f(x) - F(x)| \leq \delta(x)$ for $x \in [a, b]$ and m, the number of linear segments of F, is a minimum.

THE ALGORITHM

We consider each $f(x_i)$ as the center of a vertical line segment of length $2\delta_i$ over the corresponding x_i. We denote by $L_t(x)$ and $L_b(x)$ the continuous piecewise linear functions $f + \delta$ and $f - \delta$ respectively.

Briefly our algorithm successively determines the longest "line of sight" into the two dimensional "tunnel" bounded by L_b and L_t. The process terminates since there are only a finite number of data points.

We index the approximating line segments by a parameter k initially set to 1. We define a parameter $\lambda(k)$ taking values $+1$ and -1 which indicates whether the previous line segment, the $(k-1)$th, terminated on L_t, $\lambda(k) = +1$, or on L_b, $\lambda(k) = -1$. It is evident that any line segment which at each point x of its domain, is within $\delta(x)$ of $f(x)$ must either extend to $x = b$ or terminate either on L_t or on L_b. Initially we arbitrarily take $\lambda(1) = +1$.

We are interested in the maximum and minimum permissible slopes for an approximating line segment through a given point. Accordingly we define:

$$S_k(x, y) = [L_t(y) - f(x) - \lambda(k)\delta(x)]/(y - x), \quad (2.1)$$

$$s_k(x, y) = [L_b(y) - f(x) - \lambda(k)\delta(x)]/(y - x), \quad (2.2)$$

$$\Sigma_k(x, y, A) = \inf_{x < z \leq y}\{S_k(x, z), A\},$$

and

$$\sigma_k(x, y, B) = \sup_{x < z \leq y}\{s_k(x, z), B\}.$$

$\Sigma_k(x, y, A)$ is the minimum of the maximum permissible slopes of approximating segments over $[x, y]$ through $(x, f(x) + \lambda(k)\delta(x))$ which are not greater than some given slope A. Similarly $\sigma_k(x, y, B)$ is the maximum of the minimum permissible slopes of approximating segments over $[x, y]$ through $(x, f(x) + \lambda(k)\delta(x))$ which are not less than some given slope B.

If $\Sigma_k(x, y, A) \geq \sigma_k(x, y, B)$, then an approximating segment can be drawn through $(x, f(x) + \lambda(k)\delta(x))$ which extends at least from x to y. We are naturally interested in the largest y for which this is true. We define

$$Y_k(x, A, B) = \sup\{y \mid y \leq b, \Sigma_k(x, y, A) \geq \sigma_k(x, y, B)\},$$

$$H_k(x, A, B) = \max\{i \mid x_i \in P, x_i \leq Y_k(x, A, B)\}.$$

$Y_k(x, A, B)$ is the right endpoint of the domain of the longest approximating segment through $(x, f(x) + \lambda(k)\delta(x))$ with slope between B and A. The value of $Y_k(x, A, B)$ may not be in P, but this value can be determined since f is a continuous piecewise linear function. We will return to this point.

Before giving the algorithm we define three more quantities of interest. Let

$$X_k(x, A, B) = \begin{cases} \max[x, \{x_i \mid x_i \in P, x_i < Y_k(x, A, B), \\ S_k(x, x_i) = \Sigma_k(x, x_i, A)\}] \quad \text{if} \quad \lambda(k) = +1 \\ \max[x, \{x_i \in P, x_i < Y_k(x, A, B), \\ s_k(x, x_i) = \sigma_k(x, x_i, B)\}] \quad \text{if} \quad \lambda(k) = -1, \end{cases}$$

(the sets defined in braces in the previous expression may be empty),

$$P_k(w, x, y, z) = \begin{cases} \min\{\min[s_k(y, x_i) \mid x_i \in P, w \leq x_i < y], \\ \min[S_k(y, x_i) \mid x_i \in P, y < x_i \leq z]\} \\ \text{if} \quad \lambda(k) = +1, \\ s_k(x, y) \quad \text{if} \quad \lambda(k) = -1, \end{cases}$$

$$Q_k(w, x, y, z) = \begin{cases} S_k(x, y) \quad \text{if} \quad \lambda(k) = +1, \\ \max\{\max[S_k(y, x_i) \mid x_i \in P, w \leq x_i < y], \\ \max[x_k(y, x_i) \mid x_i \in P, y < x_i \leq x]\} \\ \text{if} \quad \lambda(k) = -1. \end{cases}$$

$X_k(x, A, B)$ determines a "pivot point" in our two dimensional tunnel to the right of x. This will be taken as a new "eyepoint" for determining a new "line of sight" extending farther to the right. P_k and Q_k determine respectively a maximum and a minimum slope for a new segment through the pivot point. These will determine the parameters A and B for subsequent evaluations of σ_k, Σ_k, Y_k, X_k. This is another point to which we shall return.

We are now ready to give our algorithm:

Initialization:

Set: $z_0 = z^* = u_0 = a$, $\lambda(1) = +1$, $j = 1$, $k = 1$,

$$A_0 = [f(x_1) + \delta(x_1) - f(x_0) - \delta(x_0)]/(x_1 - x_0),$$

$$B_0 = [f(x_1) - \delta(x_1) - f(x_0) - \delta(x_0)]/(x_1 - x_0).$$

Set an indicator if the approximation is to be continuous.

Iteration:

1. Compute: $y_j = Y_k(z_{j-1}, A_{j-1}, B_{j-1})$,

   $h = H_k(z_{j-1}, A_{j-1}, B_{j-1}) + 1$,

   $z_j = X_k(z_{j-1}, A_{j-1}, B_{j-1})$.
2. If $y_j = b$, then go to Termination step 4.
3. If $\lambda(k) = +1$ and $S_k(z_{j-1}, x_h) < \sigma_k(z_{j-1}, x_h, B_{j-1})$, or if $\lambda(k) = -1$ and $s_k(z_{j-1}, x_h) > \Sigma_k(z_{j-1}, x_h, A_{j-1})$, then go to Termination step 1.
4. If $z_j = z_{j-1}$, then go to Termination step 2.
5. Compute: $A_j = P_k(z^*, z_{j-1}\ z_j, y_j)$,

   $B_j = Q_k(z^*, z_{j-1}, z_j, y_j)$.
6. Increment j by $+1$ and go to Iteration step 1.

Termination:

1. If $\lambda(k) = +1$, then set $p_k = S_k(z_j, y_j)$. If $\lambda(k) = -1$, then set $p_k = s_k(z_j, y_j)$. Set $q_k = f(z_j) + \lambda(k)\delta(z_j) - p_k z_j$. Set $\lambda(k+1) = \lambda(k)$. Go to Termination step 3.
2. If $\lambda(k) = -1$, then set $p_k = s_k(z_j, y_j)$. If $\lambda(k) = +1$, then set $p_k = S_k(z_j, y_j)$. Set $q_k = f(z_j) + \lambda(k)\delta(z) - p_k z_j$. Set $\lambda(k+1) = -\lambda(k)$.
3. Set: $u_k = y_j$,

   $z_0 = y_j$,

   $A_0 = S_k(z_0, x_h)$,

   $B_0 = s_k(z_0, x_h)$.

   If the approximation is to be continuous, then set $z^* = z_j$, otherwise set $z^* = y_j$. If the approximation is to be continuous and $k > 1$, then recompute u_{k-1}, the abscissa value at the intersection of the current and previous segments. Set: $j = 1$, increment k by $+1$ and go to Iteration step 1.
4. Set: $p_k = \{\Sigma_k(z_{j-1}, b, A_{j-1}) + \sigma_k(z_{j-1}, b, B_{j-1})\}/2$,

   $q_k = f(z_{j-1}) + \lambda(k)\delta(z_{j-1}) - p_k z_{j-1}$,

   $u_k = b$.

   If the approximation is to be continuous and $k > 1$, then recompute u_{k-1}, the abscissa value at the intersection of the current and previous segments. End.

VALIDATION

In this section we show that $Y_k(x, A, B)$ can be computed, that the algorithm just given does generate a piecewise linear approximation which is extremal in the sense already defined and finally that the modification to be implemented for a continuous approximation does generate an extremal continuous piecewise linear approximation as asserted.

Lemma 1:

For a fixed x the functions $S_k(x, y)$ and $s_k(x, y)$ are continuous piecewise linear functions of $(y-x)^{-1}$ for $y \in (x, b]$, and the partition of the functions into segments linear in $(y-x)^{-1}$ is exactly P.

Proof:

The functions are rational and hence continuous on $(x, b]$. Let x_i and x_{i+1} be successive points of $P \cap [x, b]$. That S_k and s_k are linear in $(y-x)^{-1}$ for $y \in (x_i, x_{i+1}]$ follows from substituting

$$L_\mu(y) = L_\mu(x_i) + [L_\mu(x_{i+1}) - L_\mu(x_i)](y - x_i)/(x_{i+1} - x_i)$$

for $\mu = t$ into the definition of S_k equation (2.1) and for $\mu = b$ into the definition of s_k equation 2.2. Writing $(y - x_i) = (y - x) + (x - x_i)$ and dividing out $(y-x)$ gives an expression linear in $(y-x)^{-1}$.

Corollary 1:

For a fixed $x \in [a, b]$, if x_i and x_{i+1} are successive points of $P \cap [x, b]$, then $S_k(x, y)$ and $s_k(x, y)$ are monotone functions of y for $y \in (x_i, x_{i+1}]$.

Corollary 2:

For fixed $x \in [a, b]$, $S_k(x, y)$ and $s_k(x, y)$ take their minimum and maximum values respectively at points y' and y'' which belong to $P \cap (x, b]$.

Lemma 2:

For a fixed $x \in [a, b]$, A and B, if x_i and x_{i+1} are successive points of $P \cap (x, b]$ with $\Sigma_k(x, x_i, A) \geq \sigma_k(x, x_i, B)$ and $\Sigma_k(x, x_{i+1}, A) < \sigma_k(x, x_{i+1}, B)$, then either $s_k(x, x_{i+1}) > \Sigma_k(x, x_i, A)$ or $S_k(x, x_{i+1}) < \sigma_k(x, x_i, B)$ and not both.

Proof:

We show first that under the hypothesis not both inequalities may hold. Since $S_k(x, x_{i+1}) \geq s_k(x, x_{i+1})$, both inequalities would imply $\sigma_k(x, x_i, B) > \Sigma_k(x, x_i, A)$ which contradicts the hypothesis.

We now show that under the hypothesis not both inequalities may fail. From the definition of Σ_k and σ_k, and lemma 1: If $s_k(x, x_{i+1}) \leq \Sigma_k(x, x_i, A)$ and $\sigma_k(x, x_i, B) \leq \Sigma_k(x, x_i, A)$, then $\sigma_k(x, x_{i+1}, B) \leq \Sigma_k(x, x_i, A)$; similarly if $S_k(x, x_{i+1}) \geq \sigma_k(x, x_i, B)$ and $\Sigma_k(x, x_i, A) \geq \sigma_k(x, x_i, B)$, then $\Sigma_k(x, x_{i+1}\ A) \geq \sigma_k(x\ x_i, B)$. Thus, if the hypothesis held but both inequalities failed we would have: $\Sigma_k(x, x_i, A) \geq \sigma_k(x, x_{i+1}, B) > \Sigma_k(x, x_{i+1}, A) \geq \sigma_k(x, x_i, B)$. But this implies $s_k(x, x_{i+1}) > S_k(x, x_{i+1})$ which is impossible.

Lemma 3:

For a fixed $x\in[a, b]$, A and B, let x_i and x_{i+1} be successive points of $P\cap(x, b]$ with $\Sigma_k(x, x_i, A)\geq \sigma_k(x, x_i, B)$ and $\sigma_k(x, x_{i+1}, B)>\Sigma_k(x, x_{i+1}, A)$. If $s_k(x, x_{i+1})>\Sigma_k(x, x_i, A)$, then $Y_k(x, A, B)$ is the z value for which

$$L_b(z)=(z-x)\Sigma_k(x, x_i, A)+f(x)+\lambda(k)\delta(x);$$

while if $S_k(x, x_{i+1})<\sigma_k(x, x_i, B)$, then $Y_k(x, A, B)$ is the z value for which

$$L_t(z)=(z-x)\sigma_k(x, x_i, B)+f(x)+\lambda(k)\delta(x).$$

Proof:

Lemma 2 shows that one of these formulae can be selected. The formulae follow from the definitions of S_k and s_k, Equations (2.1) and (2.2) and the observations that if $s_k(x, x_{i+1})>\Sigma_k(x, x_i, A)$, then $s_k(x, z)$ is an increasing function of z for $z\in(x_i, x_{i+1}]$ and we are interested in the z for which $s_k(x, z)=\Sigma_k(x, x_i, A)$, while if $S_k(x, x_{i+1})<\sigma_k(x, x_i, B)$, then $S_k(x, z)$ is a decreasing function of z for $z\in(x_i, x_{i+1}]$ and we are interested in the z for which $S_k(x, z)=\sigma_k(x, x_i, B)$.

This lemma shows constructively that $Y_k(x, A, B)$ can be computed. This is the only function we have defined about which there was potential doubt.

It is evident that our algorithm generates a piecewise linear approximation which is within δ of f. We now show that this approximation is extremal.

For convenience we identify the last z_j associated with the kth segment as e_k, the abscissa value of the eyepoint of this segment. This value is available at the termination step of each segment. Note that

$$f(e_k)+\lambda(k)\delta(e_k)=e_kp_k+q_k.$$

Since each segment, except possibly the right most, terminates on either L_t or L_b we have shown that each approximating segment, again except possibly the right most, has at least two points on the boundary of our two dimensional tunnel. The extremal property of our approximation follows from each approximating segment, except possibly the right most, having at least three points on the boundary which alternate between the boundary lines.

We will prove this assertion in the next two lemmas. The first of these merely establishes a useful geometric fact.

Lemma 4:

Let $a\leq x<y<z$, and let R stand for any of: $>$, $<$, $\geq$, $\leq$, or $=$. If $\lambda(k)=+1$ and $S_k(x, y)Rs_k(x, z)$, then $S_k(x, y)Rs_k(y, z)$. If $\lambda(k)=-1$ and $s_k(x, y)RS_k(x, z)$, then $s_k(x, y)RS_k(y, z)$.

Proof:

If $\lambda(k)=+1$, then the hypotheses imply:

$$[L_t(y)-L_t(x)](z-x)R[L_b(z)-L_t(x)](y-x).$$

In this relation we substitute $(z-y+y-x)$ for $(z-x)$ on the left and $[L_b(z)-L_t(y)+L_t(y)-L_t(x)]$ for $[L_b(z)-L_t(x)]$ on the right. The result is:

$$[L_t(y)-L_t(x)](z-y)R[L_b(z)-L_t(y)](y-x).$$

The conclusion, $S_k(x, y)Rs_k(y, z)$, follows from dividing this relation by $(z-y)(y-x)>0$.

The argument is similar in the case $\lambda(k)=-1$.

Lemma 5:

If the kth approximating segment is not the right most and terminates on L_μ, where $\mu=b$ or t, then there exist $x'\in[u_{k-1}, u_k)$ and $x''\in(x', u_k)$ such that $L_\mu(x')=x'p_k+\lambda(k)\delta(x')$ and $L_v(x'')=x''p_k+\lambda(k)\delta(x'')$ where if $\mu=b$, then $v=t$ and vice versa.

Proof:

The idea of the proof is simple although the notation is complicated. The outline of the proof is as follows. If the kth segment is not the right most, then the slope of this segment is both the maximum of the minimum permissible slopes for approximating segments through, $(e_k, L_\mu(e_k))$ which intersect the vertical segment over u_{k-1}, and the minimum of the maximum permissible slopes for approximating segments through this eyepoint which intersect this vertical segment. Furthermore each of these slopes is determined by a point on the boundary. One of these points is the terminal point of the segment, the other is the point whose existence this lemma asserts.

There are four possible cases determined by: $L_t(u_k)=u_kp_k+q_k$ with $\lambda(k)=\pm1$, and $L_b(u_k)=u_kp_k+q_k$ with $\lambda(k)=\pm1$. We shall consider the two cases with $\lambda(k)=+1$. The other two are similar to these.

We consider first $\lambda(k)=+1$ and $L_t(u_k)=u_kp_k+q_k$. Then $L_t(e_k)=e_kp_k+q_k$, $x'=e_k$ and we need to find $x''\in(x', u_k)$. If $e_k=u_{k-1}$, then the slope of the line through $(u_{k-1}, L_t(u_{k-1}))$ and $(u_{k+1}, L_t(u_{k+1}))$ is the minimum permissible slope of an approximating segment over $[u_{k-1}, u_k]$. By lemma 3 and corollary 2 of lemma 1 this minimum slope is the slope of a line from $(u_{k-1}, L_t(u_{k-1}))$ through $(x_i, L_b(x_i))$ for some $x_i\in P\cap(u_{k-1}, u_k]$. Furthermore, this x_i is not u_k since $L_t(u_k)=u_kp_k+q_k$. In this case x'' is this x_i.

If $e_k\neq u_{k-1}$, then to determine u_k, p_k and q_k more than one pass through the iteration phase of our algorithm was required. Let e_{k^*} and y_{k^*} be respectively z_{j-2} and y_{j-1}, where $e_k=z_j$ from the determination of u_k, p_k and

q_k and let $x_{k*}=\min\{x \mid x\in P, x\geq y_{k*}\}$. Then

$$p_k=\Sigma_k(e_k, u_k, A)=\sigma_k(e_k, u_k, B),$$

where

$$A=P_k(u_{k-1}, e_{k*}, e_k, y_{k*}),$$

and

$$B=Q_k(u_{k-1}, e_{k*}, e_k, y_{k*}).$$

If there were no $x''\in(e_k, u_k)$ such that $L_b(x'')=x''p_k+q_k$, then p_k would equal B and $s_k(e_k, x_i)$ would be less than B for each $x_i\in P\cap(e_k, u_k]$. But this is impossible. For the algorithm which gives the value of y_j insures that $S_k(e_{k*}, e_k)\leq s_k(e_{k*}, x_{k*})$; and lemma 4 then implies $S_k(e_{k*}, e_k)\leq s_k(e_k, x_{k*})$. Finally, since $\lambda=+1$, we have $B=S_k(e_{k*}, e_k)$. Hence $s_k(e_k, x_{k*})\geq B$, and the existence of an x'' is established.

We suppose now that $\lambda(k)=+1$ and $L_b(u_k)=u_kp_k+q_k$. Then $L_t(e_k)=e_kp_k+q_k$ and we will show that there exists $x'\in[u_{k-1}, e_k)$ such that $L_b(x')=x'p_k+q_k$. This follows from $s_k(e_k, u_k)=P_k(u_{k-1}, e_{k*}, e_k, y_{k*})$. For $s_k(e_k, u_k)=\sigma_k(e_k, u_k, B)=\Sigma_k(e_k, u_k, A)$, and if this is not A, namely $P_k(u_{k-1}, e_{k*}, e_k, y_{k*})$, then there exists some $x_i\in P\cap(e_k, u_k)$ such that $S_k(e_k, x_i)<A$. But this contradicts the definition of e_k as the last z_j from the iteration phase in the determination of u_k, p_k and q_k.

Corollary 1:

There is no approximating linear segment whose domain includes u_{k-1} which extends farther to the right than u_k.

Proof:

Such a segment would have to cross our kth approximating segment twice.

Corollary 2:

Our approximation is extremal in the sense stated.

Proof:

Any other linear approximation of fewer line segments must have at least one segment whose domain begins to the left of and ends to the right of the domain of one of our approximating segments.

Lemma 6:

The modification of our algorithm for a continuous approximation does give a continuous piecewise linear approximation which is extremal in the sense we have defined.

Proof:

The modification is to one of the argument values where P_k and Q_k are to be evaluated. The result is to insure that for $k>1$ the kth segment extended to the left intersects the $(k-1)$th segment between the x'' and u_k of lemma 5. That this approximation is extremal follows from the observation made in the proof of corollary 2 of lemma 5.

SAMPLE RESULTS

The appended FORTRAN program, with a suitable driver, was run a number of times. An input parameter determined if the approximation was to be continuous and whether or not the deviation was to vary. If the deviation was not to vary it was taken as the first input tolerance.

One set of test data was the following:

```
  INPUT DATA, NUMBER OF POINTS=20

      X            Y         TOLERANCE

   1.0000       0.7175        0.1250
   2.0000       0.6250        0.0625
   3.0000       0.6250        0.0625
   4.0000       0.6250        0.1250
   5.0000       0.6563        0.1250
   6.0000       0.7500        0.2500
   7.0000       0.6563        0.1250
   8.0000       0.6406        0.0313
   9.0000       0.7500        0.0312
  10.0000       0.5000        0.0156
  11.0000       0.7500        0.1250
  12.0000       0.6250        0.0625
  13.0000       0.6250        0.0625
  14.0000       0.6250        0.1250
  15.0000       0.6563        0.1250
  16.0000       0.7500        0.2500
  17.0000       0.6563        0.1250
  18.0000       0.6406        0.0313
  19.0000       0.7500        0.0312
  20.0000       1.0000        0.0156
```

The results, when the deviation was taken as constant, namely 0.125, were:

```
2 LINE SEGMENTS WERE REQUIRED TO
     FIT THESE DATA EQUATION OF
     APPROXIMATION IS  Y=PX+Q
          FOR U.LT.X.LT.V

   P        Q        U       V      Y(U)   Y(V)

 0.0      0.6250  1.0000 19.0000 0.6250 0.6250
 0.3750  -6.5000 19.0000 20.0000 0.6250 1.0000
```

whether or not the approximation was required to be continuous. When the deviation was permitted to vary, the results were:

```
6 LINE SEGMENTS WERE REQUIRED TO
    FIT THESE DATA EQUATION OF
    APPROXIMATION  IS  Y=PX+Q
         FOR U.LT.X.LT.V

    P         Q        U        V     Y(U)   Y(V)

  0.0113   0.5812   1.0000   8.6372 0.5925 0.6791
 -0.0627   1.2828   8.6372   9.3075 0.7415 0.6995
 -0.1893   2.4086   9.3075  10.0947 0.6467 0.4977
  0.0832  -0.2904  10.0947  11.9235 0.5497 0.7018
  0.0172   0.3621  11.9235  18.6778 0.5673 0.6835
  0.2393  -3.7867  18.6778  20.0000 0.6835 1.0000
```

when the approximation was not required to be continuous, and:

```
6 LINE SEGMENTS WERE REQUIRED TO
    FIT THESE DATA EQUATION OF
    APPROXIMATION  IS  Y=PX+Q
         FOR U.LT.X.LT.V

    P         Q        U        V     Y(U)   Y(V)

  0.0113   0.5812   1.0000   8.0000 0.5925 0.6719
  0.0469   0.2965   8.0000   9.0000 0.6719 0.7188
 -0.2032   2.5474   9.0000  10.0000 0.7188 0.5156
  0.1094  -0.5781  10.0000  11.0000 0.5156 0.6250
  0.0067   0.5513  11.0000  18.6084 0.6250 0.6759
  0.2329  -3.6572  18.6084  20.0000 0.6759 1.0000
```

when the approximation was required to be continuous.

These results show that it may not require any additional segments to have the approximation be continuous for a given set of tolerances.

Another set of test data was:

```
  INPUT DATA, NUMBER OF POINTS=13

     X            Y        TOLERANCE

   0.0          5.0000       0.2000
   1.0000       4.0000       0.1000
   2.0000       3.1000       0.3000
   3.0000       3.1000       0.0100
   4.0000       3.1000       0.0030
   5.0000       4.0000       0.2000
   6.0000       5.0000       0.3000
   7.0000       6.0000       0.5000
   8.0000       6.0000       0.2000
   9.0000       6.0000       0.1000
  10.0000       4.0000       0.7500
  11.0000       2.5000       1.0000
  12.0000       1.0000       0.2000
```

The results were:

```
5 LINE SEGMENTS WERE REQUIRED TO
    FIT THESE DATA EQUATION OF
    APPROXIMATION  IS  Y=PX+Q
         FOR U.LT.X.LT.V

    P         Q        U        V     Y(U)   Y(V)

 -0.7500   4.8000   0.0      2.5333 4.8000 2.9000
  0.2727   2.2091   2.5333   4.0000 2.9000 3.3000
  0.8333  -0.0333   4.0000   7.4800 3.3000 6.2000
 -0.2632   8.1684   7.4800   9.0360 6.2000 5.7905
 -1.6500  20.7000   9.0360  12.0000 5.7905 0.9000
```

when the approximation was required to be continuous and the deviation was taken as 0.2; and:

```
5 LINE SEGMENTS WERE REQUIRED TO
    FIT THESE DATA EQUATION OF
    APPROXIMATION  IS  Y=PX+Q
         FOR U.LT.X.LT.V

    P         Q        U        V     Y(U)   Y(V)

 -0.7000   4.8000   0.0      2.4530 4.8000 3.0829
  0.0130   3.0510   2.4530   4.0000 3.0829 3.1030
  0.7990  -0.0930   4.0000   7.9099 3.1030 6.2270
 -0.3000   8.6000   7.9099   9.1905 6.2270 5.8429
 -1.7237  21.6847   9.1905  12.0000 5.8428 1.0000
```

when the approximation was required to be continuous and the deviation was permitted to vary.

The program will accept any non-negative tolerance including zero. If all tolerances are zero the program gives the parameters for linear interpolation.

ACKNOWLEDGMENTS

This problem was originally given to me by Mr. Fred Stahl of the John D. Kettelle Corporation. I am indebted to Dr. W. A. Thedford of Virginia Commonwealth University for a suggestion which led to the modification for a continuous approximation.

Mr. W. T. Lipscomb III, a student at V.C.U., programmed a preliminary version of the algorithm and in numerous discussions contributed some of the descriptive terminology. His name appears as a coauthor of the appended listing.

Sincere appreciation goes to my wife Gwendolyn who typed this manuscript.

REFERENCES

1 R BELLMAN
On the approximation of curves by line segments using dynamic programming
Comm ACM 4 6 June 1961 p 284

2 B GLUSS
Further remarks on line segment curve-fitting using dynamic programming
Comm ACM 5 8 August 1962 pp 441-443

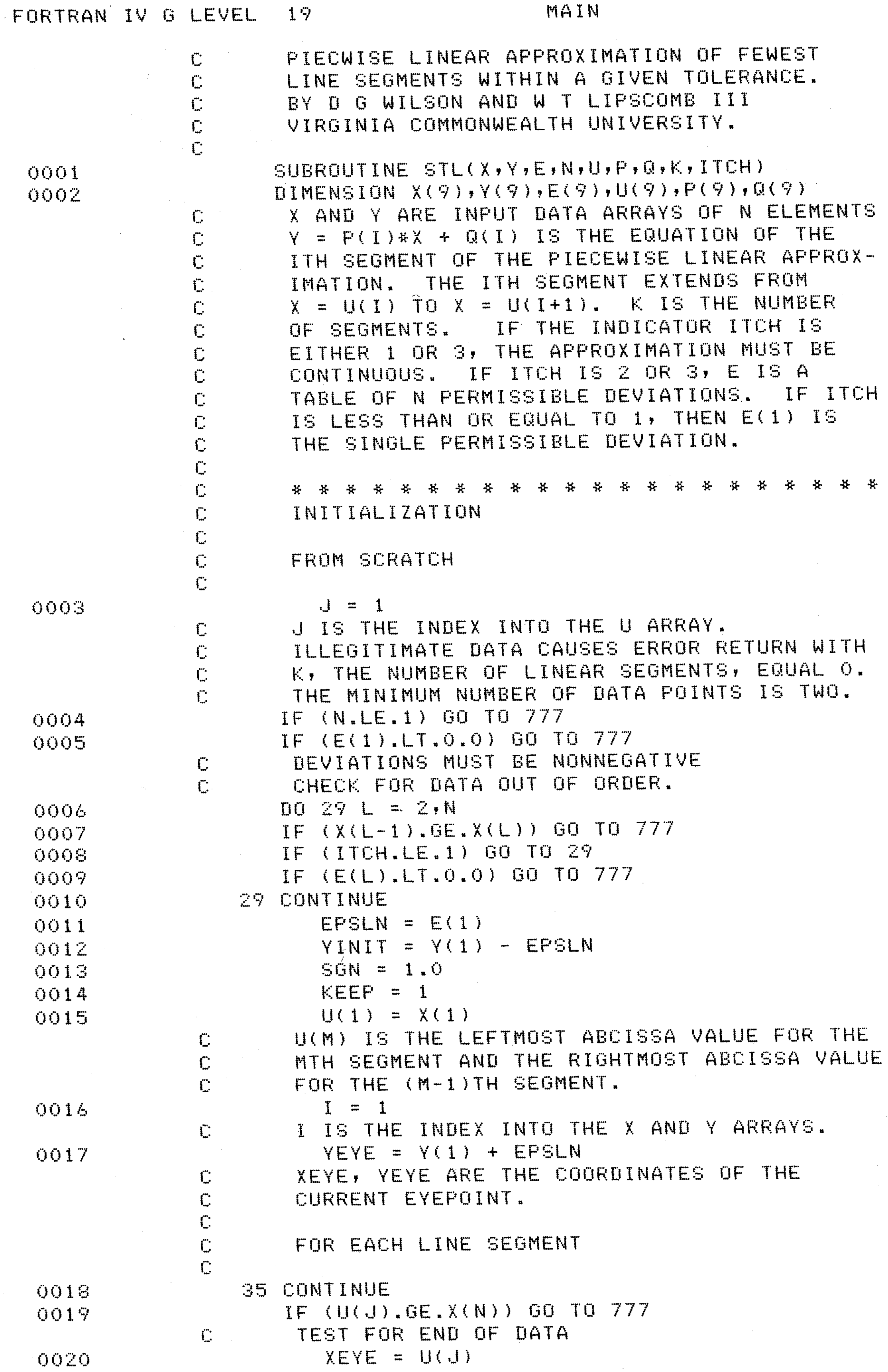
3 G M PHILLIPS
Algorithms for piecewise straight line approximations
The Computer J 11 2 August 1968 pp 211-212

4 H SCHWERDTFEGER
Interpolation and curve fitting by sectionally linear functions
Canad Math Bull 3 1 January 1960 pp 41-57

5 ———
Further remarks on sectionally linear functions
Canad Math Bull 4 1 January 1961 pp 53-55

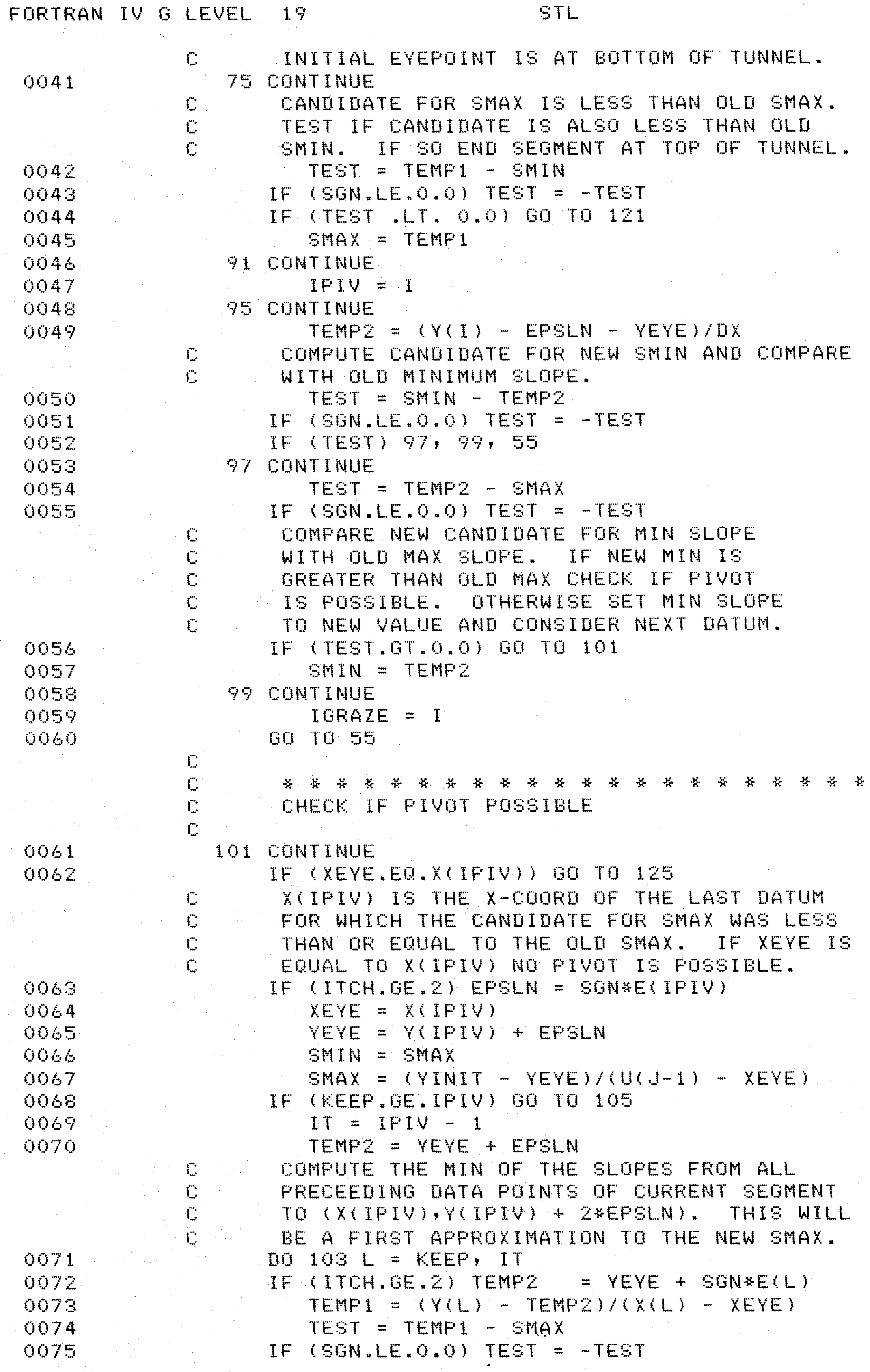
6 H STONE
Approximation of curves by line segments
Math of Comp 15 73 January 1961 pp 40-47

APPENDIX



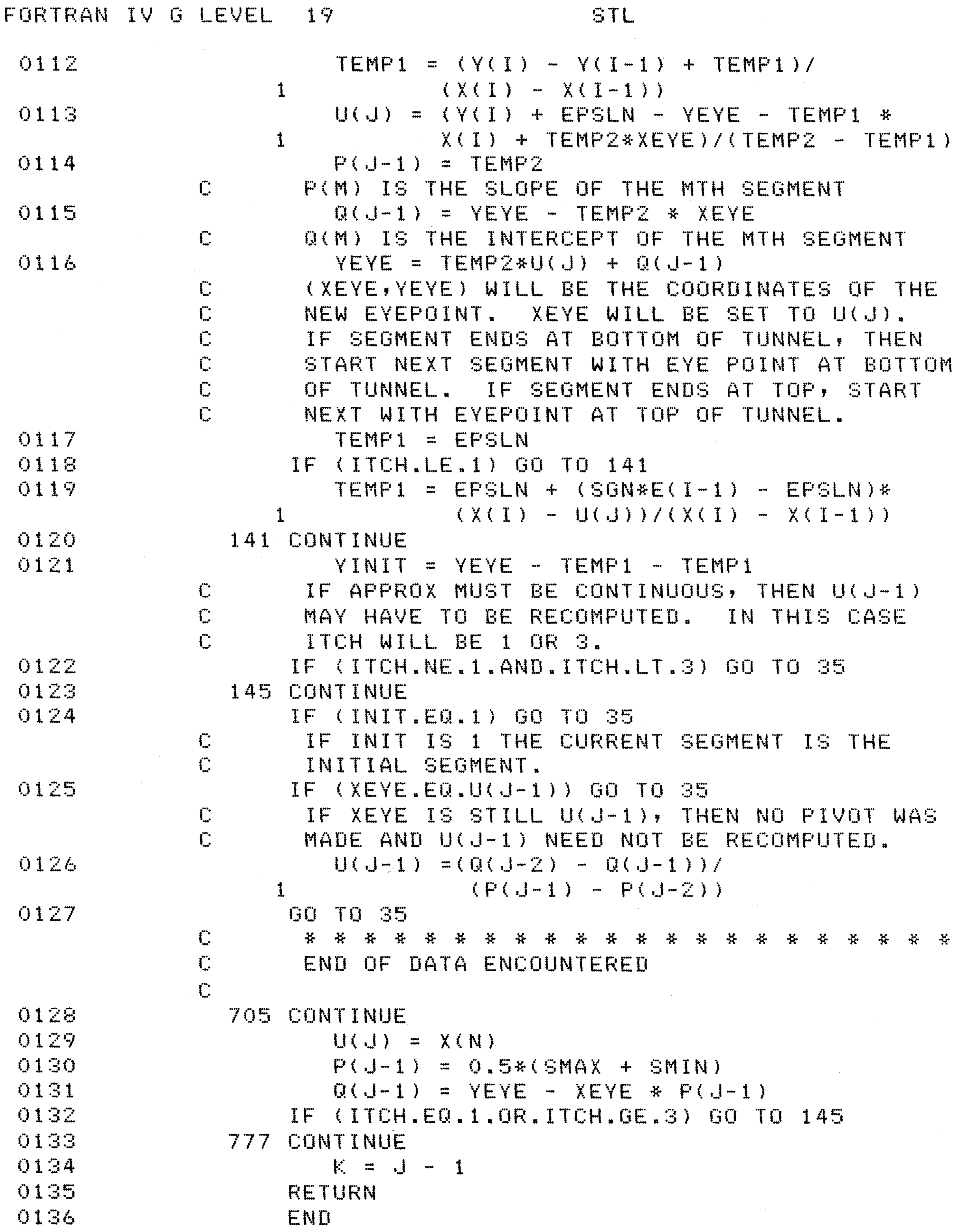
```
 FORTRAN IV G LEVEL  19                     MAIN

                   C      PIECWISE LINEAR APPROXIMATION OF FEWEST
                   C      LINE SEGMENTS WITHIN A GIVEN TOLERANCE.
                   C      BY D G WILSON AND W T LIPSCOMB III
                   C      VIRGINIA COMMONWEALTH UNIVERSITY.
                   C
 0001                    SUBROUTINE STL(X,Y,E,N,U,P,Q,K,ITCH)
 0002                    DIMENSION X(9),Y(9),E(9),U(9),P(9),Q(9)
                   C      X AND Y ARE INPUT DATA ARRAYS OF N ELEMENTS
                   C      Y = P(I)*X + Q(I) IS THE EQUATION OF THE
                   C      ITH SEGMENT OF THE PIECEWISE LINEAR APPROX-
                   C      IMATION.  THE ITH SEGMENT EXTENDS FROM
                   C      X = U(I) TO X = U(I+1).  K IS THE NUMBER
                   C      OF SEGMENTS.   IF THE INDICATOR ITCH IS
                   C      EITHER 1 OR 3, THE APPROXIMATION MUST BE
                   C      CONTINUOUS.  IF ITCH IS 2 OR 3, E IS A
                   C      TABLE OF N PERMISSIBLE DEVIATIONS.  IF ITCH
                   C      IS LESS THAN OR EQUAL TO 1, THEN E(1) IS
                   C      THE SINGLE PERMISSIBLE DEVIATION.
                   C
                   C      * * * * * * * * * * * * * * * * * * * * * *
                   C      INITIALIZATION
                   C
                   C      FROM SCRATCH
                   C
 0003                       J = 1
                   C      J IS THE INDEX INTO THE U ARRAY.
                   C      ILLEGITIMATE DATA CAUSES ERROR RETURN WITH
                   C      K, THE NUMBER OF LINEAR SEGMENTS, EQUAL 0.
                   C      THE MINIMUM NUMBER OF DATA POINTS IS TWO.
 0004                    IF (N.LE.1) GO TO 777
 0005                    IF (E(1).LT.0.0) GO TO 777
                   C      DEVIATIONS MUST BE NONNEGATIVE
                   C      CHECK FOR DATA OUT OF ORDER.
 0006                    DO 29 L = 2,N
 0007                    IF (X(L-1).GE.X(L)) GO TO 777
 0008                    IF (ITCH.LE.1) GO TO 29
 0009                    IF (E(L).LT.0.0) GO TO 777
 0010                 29 CONTINUE
 0011                       EPSLN = E(1)
 0012                       YINIT = Y(1) - EPSLN
 0013                       SGN = 1.0
 0014                       KEEP = 1
 0015                       U(1) = X(1)
                   C      U(M) IS THE LEFTMOST ABCISSA VALUE FOR THE
                   C      MTH SEGMENT AND THE RIGHTMOST ABCISSA VALUE
                   C      FOR THE (M-1)TH SEGMENT.
 0016                       I = 1
                   C      I IS THE INDEX INTO THE X AND Y ARRAYS.
 0017                       YEYE = Y(1) + EPSLN
                   C      XEYE, YEYE ARE THE COORDINATES OF THE
                   C      CURRENT EYEPOINT.
                   C
                   C      FOR EACH LINE SEGMENT
                   C
 0018                 35 CONTINUE
 0019                    IF (U(J).GE.X(N)) GO TO 777
                   C      TEST FOR END OF DATA
 0020                       XEYE = U(J)
```

```
FORTRAN IV G LEVEL  19                  STL
             C      U(J-1), YINIT ARE THE COORDINATES OF
             C      THE POINT OPPOSITE THE ORIGONAL EYEPOINT
             C      OF THE CURRENT SEGMENT.
 0021                 INIT = I
             C      INIT IS THE INDEX OF THE FIRST INPUT
             C      X VALUE IN THE CURRENT SUBDIVISION.
             C      THIS MAY NOT BE THE INITIAL X-VALUE
             C      OF THE CURRENT SEGMENT.
 0022              IF (XEYE .GE. X(I)) I = I + 1
             C      TEST FOR SEGMENT ENDING AT AN INPUT X VALUE
 0023              IF (ITCH.NE.1.AND.ITCH.LT.3) KEEP = INIT
 0024              IF (ITCH.GE.2) EPSLN = SGN*E(I)
 0025                 DX = X(I) - XEYE
             C      DX = DISTANCE FROM CURRENT EYEPOINT TO X-
             C      COORDINATE OF POINT CURRENTLY BEING CONSID-
             C      ERED.  TO BE USED IN COMPUTING MAX AND MIN
             C      SLOPES SMAX AND SMIN RESPECTIVELY
 0026                 SMAX = (Y(I) + EPSLN - YEYE)/DX
 0027                 SMIN = (Y(I) - EPSLN - YEYE)/DX
 0028                 IPIV = I
             C      IPIV IS THE INDEX OF THE RIGHTMOST DATA
             C      POINT FOR WHICH THE CANDIDATE FOR SMAX IS
             C      LESS THAN OR EQUAL TO THE CURRENT SMAX.
             C      THIS LOCATES THE CANDIDATE FOR NEW EYEPOINT
             C      IF PIVOTING IS NECESSARY.
 0029                 IGRAZE = I
             C      IGRAZE IS THE INDEX OF THE RIGHTMOST DATA
             C      POINT FOR WHICH THE CANDIDATE FOR SMIN IS
             C      GREATER THAN OR EQUAL TO THE CURRENT SMIN
 0030                 J = J + 1
             C      INCREMENT INDEX INTO OUTPUT ARRAYS
             C
             C      END OF INITIALIZATION
             C      * * * * * * * * * * * * * * * * * * * * * *
             C      DETERMINE MAX AND MIN SLOPES FOR SEGMENT
             C
 0031           55 CONTINUE
 0032              IF (I .EQ. N) GO TO 705
             C      TEST FOR END OF DATA WITHIN CURRENT SEGMENT
 0033                 I = I + 1
             C      INCREMENT INDEX INTO DATA ARRAYS
 0034           57 CONTINUE
 0035                 DX = X(I) - XEYE
             C      DX AS BEFORE
 0036              IF (ITCH.GE.2) EPSLN = SGN*E(I)
 0037                 TEMP1 = (Y(I) + EPSLN - YEYE)/DX
             C      TEMP1 IS A CANDIDATE FOR SMAX
 0038                 TEST = TEMP1 - SMAX
 0039              IF (SGN.LE.0.0) TEST = -TEST
             C      SGN WILL BE POS IF PREVIOUS SEGMENT
             C      ENDS AT TOP OF TUNNEL, NEG IF PREVIOUS
             C      SEGMENT ENDS AT BOTTOM OF TUNNEL
             C      IF SGN IS NEG CONSIDER TUNNEL TO BE UPSIDE
             C      DOWN FOR TESTS WITH TEMP1 AND TEMP2.
 0040              IF (TEST) 75, 91, 95
             C      IF CANDIDATE FOR SMAX IS GE OLD SMAX,
             C      THEN KEEP OLD SMAX.  OTHERWISE CANDIDATE
             C      BECOMES THE NEW SMAX.  EPSLN IS NEG IF
```



```
FORTRAN IV G LEVEL  19                 STL

             C      INITIAL EYEPOINT IS AT BOTTOM OF TUNNEL.
 0041           75 CONTINUE
             C      CANDIDATE FOR SMAX IS LESS THAN OLD SMAX.
             C      TEST IF CANDIDATE IS ALSO LESS THAN OLD
             C      SMIN.  IF SO END SEGMENT AT TOP OF TUNNEL.
 0042                 TEST = TEMP1 - SMIN
 0043              IF (SGN.LE.0.0) TEST = -TEST
 0044              IF (TEST .LT. 0.0) GO TO 121
 0045                 SMAX = TEMP1
 0046           91 CONTINUE
 0047                 IPIV = I
 0048           95 CONTINUE
 0049                 TEMP2 = (Y(I) - EPSLN - YEYE)/DX
             C      COMPUTE CANDIDATE FOR NEW SMIN AND COMPARE
             C      WITH OLD MINIMUM SLOPE.
 0050                 TEST = SMIN - TEMP2
 0051              IF (SGN.LE.0.0) TEST = -TEST
 0052              IF (TEST) 97, 99, 55
 0053           97 CONTINUE
 0054                 TEST = TEMP2 - SMAX
 0055              IF (SGN.LE.0.0) TEST = -TEST
             C      COMPARE NEW CANDIDATE FOR MIN SLOPE
             C      WITH OLD MAX SLOPE.  IF NEW MIN IS
             C      GREATER THAN OLD MAX CHECK IF PIVOT
             C      IS POSSIBLE.  OTHERWISE SET MIN SLOPE
             C      TO NEW VALUE AND CONSIDER NEXT DATUM.
 0056              IF (TEST.GT.0.0) GO TO 101
 0057                 SMIN = TEMP2
 0058           99 CONTINUE
 0059                 IGRAZE = I
 0060              GO TO 55
             C
             C      * * * * * * * * * * * * * * * * * * * * * *
             C      CHECK IF PIVOT POSSIBLE
             C
 0061          101 CONTINUE
 0062              IF (XEYE.EQ.X(IPIV)) GO TO 125
             C      X(IPIV) IS THE X-COORD OF THE LAST DATUM
             C      FOR WHICH THE CANDIDATE FOR SMAX WAS LESS
             C      THAN OR EQUAL TO THE OLD SMAX.  IF XEYE IS
             C      EQUAL TO X(IPIV) NO PIVOT IS POSSIBLE.
 0063              IF (ITCH.GE.2) EPSLN = SGN*E(IPIV)
 0064                 XEYE = X(IPIV)
 0065                 YEYE = Y(IPIV) + EPSLN
 0066                 SMIN = SMAX
 0067                 SMAX = (YINIT - YEYE)/(U(J-1) - XEYE)
 0068              IF (KEEP.GE.IPIV) GO TO 105
 0069                 IT = IPIV - 1
 0070                 TEMP2 = YEYE + EPSLN
             C      COMPUTE THE MIN OF THE SLOPES FROM ALL
             C      PRECEEDING DATA POINTS OF CURRENT SEGMENT
             C      TO (X(IPIV),Y(IPIV) + 2*EPSLN).  THIS WILL
             C      BE A FIRST APPROXIMATION TO THE NEW SMAX.
 0071              DO 103 L = KEEP, IT
 0072              IF (ITCH.GE.2) TEMP2    = YEYE + SGN*E(L)
 0073                 TEMP1 = (Y(L) - TEMP2)/(X(L) - XEYE)
 0074                 TEST = TEMP1 - SMAX
 0075              IF (SGN.LE.0.0) TEST = -TEST
```

```
FORTRAN IV G LEVEL  19                   STL

      IF (TEST.LT.0.0) SMAX = TEMP1
  103 CONTINUE
  105 CONTINUE
      IF (IPIV.GE.I-1) GO TO 57
         IT = I - 2
C
C      COMPUTE THE MIN OF THE SLOPES FROM ALL
C      SUCCEEDING DATA POINTS OF THE CURRENT
C      SEGMENT TO (X(IPIV),Y(IPIV)) IF THIS IS
C      LESS THAN THE PREVIOUSLY COMPUTED VALUE,
C      THEN IT BECOMES THE NEW SMAX.
         TEMP2 = YEYE - EPSLN
      DO 111 L = IPIV,IT
         DX = X(L+1) - XEYE
      IF (ITCH.GE.2) TEMP2 = YEYE - SGN*E(L+1)
         TEMP1 = (Y(L+1) - TEMP2)/DX
         TEST = TEMP1 - SMAX
      IF (SGN.LE.0.0) TEST = -TEST
      IF (TEST) 107, 109, 111
  107 CONTINUE
         SMAX = TEMP1
  109 CONTINUE
         IPIV = L  + 1
  111 CONTINUE
      GO TO 57
C
C      * * * * * * * * * * * * * * * * * * * * * *
C      END CURRENT SEGMENT
  121 CONTINUE
         TEMP2 = SMIN
         KEEP = IGRAZE
C      EQ OF THIS SEGMENT IS
C      Y = SMIN * (X - XEYE) + YEYE
C      NEW EYEPOINT WILL BE AT THE INTERSECTION
C      OF THIS LINE WITH THE LINE BETWEEN
C      (X(I-1),Y(I-1)+EPSLN) AND (X(I),Y(I)+EPSLN)
      IF (ITCH.LE.1) GO TO 135
      GO TO 129
C
  125 CONTINUE
         SGN = -SGN
         EPSLN = -EPSLN
         TEMP2 = SMAX
         KEEP = IPIV
C      EQUATION OF THIS SEGMENT IS
C      Y = SMAX * (X - XEYE) + YEYE
C      NEW EYEPOINT WILL BE AT THE INTERSECTION
C      OF THIS LINE WITH THE LINE BETWEEN
C      (X(I-1),Y(I-1)-EPSLN) AND (X(I),Y(I)-EPSLN)
      IF (ITCH.LE.1) GO TO 135
C      IF ITCH.LE.1, THEN EPSLN IS A CONSTANT
C      NAMELY E(1).
  129 CONTINUE
         TEMP1 = EPSLN - SGN*E(I-1)
      GO TO 137
  135 CONTINUE
         TEMP1 = 0.0
  137 CONTINUE
```



```
FORTRAN IV G LEVEL  19                   STL

 0112                     TEMP1 = (Y(I) - Y(I-1) + TEMP1)/
                1                (X(I) - X(I-1))
 0113                     U(J) = (Y(I) + EPSLN - YEYE - TEMP1 *
                1                X(I) + TEMP2*XEYE)/(TEMP2 - TEMP1)
 0114                     P(J-1) = TEMP2
            C           P(M) IS THE SLOPE OF THE MTH SEGMENT
 0115                     Q(J-1) = YEYE - TEMP2 * XEYE
            C           Q(M) IS THE INTERCEPT OF THE MTH SEGMENT
 0116                     YEYE = TEMP2*U(J) + Q(J-1)
            C           (XEYE,YEYE) WILL BE THE COORDINATES OF THE
            C           NEW EYEPOINT.  XEYE WILL BE SET TO U(J).
            C           IF SEGMENT ENDS AT BOTTOM OF TUNNEL, THEN
            C           START NEXT SEGMENT WITH EYE POINT AT BOTTOM
            C           OF TUNNEL.  IF SEGMENT ENDS AT TOP, START
            C           NEXT WITH EYEPOINT AT TOP OF TUNNEL.
 0117                     TEMP1 = EPSLN
 0118                  IF (ITCH.LE.1) GO TO 141
 0119                     TEMP1 = EPSLN + (SGN*E(I-1) - EPSLN)*
                1                 (X(I) - U(J))/(X(I) - X(I-1))
 0120             141 CONTINUE
 0121                     YINIT = YEYE - TEMP1 - TEMP1
            C           IF APPROX MUST BE CONTINUOUS, THEN U(J-1)
            C           MAY HAVE TO BE RECOMPUTED.  IN THIS CASE
            C           ITCH WILL BE 1 OR 3.
 0122                  IF (ITCH.NE.1.AND.ITCH.LT.3) GO TO 35
 0123             145 CONTINUE
 0124                  IF (INIT.EQ.1) GO TO 35
            C           IF INIT IS 1 THE CURRENT SEGMENT IS THE
            C           INITIAL SEGMENT.
 0125                  IF (XEYE.EQ.U(J-1)) GO TO 35
            C           IF XEYE IS STILL U(J-1), THEN NO PIVOT WAS
            C           MADE AND U(J-1) NEED NOT BE RECOMPUTED.
 0126                     U(J-1) =(Q(J-2) - Q(J-1))/
                1                  (P(J-1) - P(J-2))
 0127                  GO TO 35
            C           * * * * * * * * * * * * * * * * * * * * * * *
            C           END OF DATA ENCOUNTERED
            C
 0128             705 CONTINUE
 0129                     U(J) = X(N)
 0130                     P(J-1) = 0.5*(SMAX + SMIN)
 0131                     Q(J-1) = YEYE - XEYE * P(J-1)
 0132                  IF (ITCH.EQ.1.OR.ITCH.GE.3) GO TO 145
 0133             777 CONTINUE
 0134                     K = J - 1
 0135                  RETURN
 0136                  END
```

Experimental results on a new computer method for generating optimal policy variables in (s, S) inventory control problem

by P. E. VALISALO, B. D. SIVAZLIAN and J. F. MAILLOT

University of Florida
Gainesville, Florida

INTRODUCTION

The problem of developing a computable solution to the optimal periodic review stationary (s, S) inventory control problem has not received the same level of attention as the theoretical works first initiated in 1951 by the well-known paper of Arrow, Harris, Marschak.[1]

Even for some of the simplest situations, perhaps closest to practice, involving a fixed set-up cost, and a linear holding and shortage cost, the solution to the mathematical problem of determining the optimal values of s and S to minimize the total expected cost per period, requires the formal solution of an integral equation of the Volterra type and an optimization problem of a fairly complex objective function involving the decision variables s and S.

Explicit theoretical solutions for s and S can be obtained as computable formulas for some very simple demand distributions as for example the negative exponential distribution. Approximations to the optimal solutions were proposed by Roberts[2] in 1962. In 1965, Veinott and Wagner[6] developed an algorithmic procedure applicable to instances when the demand distribution is described by a discrete random variable; the study fails to discuss the computational efficiency of the algorithm and its usability in practice particularly in relation to the large number of input variables (at least five) which are necessary to compute a single pair of optimal values of (s, S). In practice, this facet of the computational problem cannot be ignored since inventory control involves usually several thousand of commodities, each commodity necessitating frequent updating of the set of input variables due to changes in economic or other conditions. In 1968, Sivazlian[3] derived a set of equations to compute the optimal values of s and $Q=S-s$ and developed conceptual approaches to solve these equations on the digital computer and on the analog computer. The feasibility of using analog computers was demonstrated in a set of experiments published in 1970 by Sivazlian and Valisalo.[5] By using dimensional analysis and using numerical inversion techniques for Laplace transforms, Sivazlian[4] generated a set of graphs for the case of gamma distribution demand with integer order, to obtain the optimal values of s and Q for all possible values of input variables. The computational feasibility of the problem was also discussed.

Although both analog and digital computer techniques proved successful, certain computational deficiencies were encountered in both methods. In using analog computers, experimental errors were generated when increasing the number of integrators. Further the derivation of results for complex demand distributions was limited by the capability of the analog computer system. In using digital computation, the discretization of the problem and the numerical method utilized could generate solution instability.

These two deficiencies were ultimately resolved by combining the conceptual basis of analog computation with the capacity and speed of digital computers by using the Continuous Simulation and Modeling Programming (C.S.M.P.) on the IBM 360/65 digital computer. The C.S.M.P. is a simulation language which works on block diagrams as an analog computer does but which utilizes a digital program. The present paper presents preliminary results obtained using this computation technique and discusses some inherent merits of this method.

THEORETICAL CONSIDERATIONS

A brief discussion of the theoretical results as obtained by Sivazlian[3] is necessary.

TABLE I

****CONTINOUS SYSTEM MODELING PROGRAM****
PROBLEM INPUT STATEMENTS

```
INITIAL
      P=10./X
      A=X/1C.
PARAM S=(0.,2.,4.,6.,8.,12.),X=19.
DYNAMIC

      XL0=0.
      XL1=REALPL(A,P,XL0)
      XL2=REALPL(0.,P,XL1)
      XL3=REALPL(0.,P,XL2)
      XL4=REALPL(0.,P,XL3)
      XL5=REALPL(0.,P,XL4)

      XL6=REALPL(0.,P,XL5)
      XL7=REALPL(0.,P,XL6)
      XL8=REALPL(0.,P,XL7)
      XL9=REALPL(0.,P,XL8)
      XL10=REALPL(0.,P,XL9)
      XL11=REALPL(0.,P,XL10)

      XL12=REALPL(0.,P,XL11)
      XL13=REALPL(0.,P,XL12)
      XL14=REALPL(0.,P,XL13)
      XL15=REALPL(0.,P,XL14)
      XL16=REALPL(0.,P,XL15)
      XL17=REALPL(0.,P,XL16)

      XL18=REALPL(0.,P,XL17)
      XL19=REALPL(0.,P,XL18)
      XLD=11.*XL19
      SM=X19-XL
      XL=INTGRL(-10.,XLD)
PROCED C=BLOC(TIME,S)

      IF(TIME-S)1,2,2
    1 C=0.
      G0 TO 3
    2 C=1.
    3 CONTINUE
ENDPRO

      SMI=C*SM
      X1=REALPL(0.,P,SMI)
      X2=REALPL(0.,P,X1)
      X3=REALPL(0.,P,X2)
      X4=REALPL(0.,P,X3)
      X5=REALPL(0.,P,X4)

      X6=REALPL(0.,P,X5)
      X7=REALPL(0.,P,X6)
      X8=REALPL(0.,P,X7)
      X9=REALPL(0.,P,X8)
      X10=REALPL(0.,P,X9)
      X11=REALPL(0.,P,X10)

      X12=REALPL(0.,P,X11)
      X13=REALPL(0.,P,X12)
      X14=REALPL(0.,P,X13)
      X15=REALPL(0.,P,X14)
      X16=REALPL(0.,P,X15)
      X17=REALPL(0.,P,X16)

      X18=REALPL(0.,P,X17)
      X19=REALPL(0.,P,X18)
      BM=INTGRL(0.,SMI)
FINISH SM=0.
```

Consider the periodic review inventory control problem of a single commodity system where the demand per period for the commodity is described by a continuous random variable $\xi(0<\xi<\infty)$ with probability density function $\phi(\xi)$ identically and independently distributed from period to period. Assume that the following cost components affect the operation of the system: a fixed order cost K, a holding cost and a shortage cost. Complete backlogging is allowed and delivery of orders is immediate. Let $L(x)$ be the expected holding and shortage cost over one period when the stock level at the beginning of the period following a decision is x. An $(\mathcal{S}, S)$ policy is in effect, that is, whenever the stock level at the beginning of a period lies between $\mathcal{S}$ and S, no order is placed, and whenever the stock level falls below $\mathcal{S}$, it is brought up to S. It can be shown[3,4] that for $\mathcal{S}\geq 0$ the values of $\mathcal{S}$ and $Q=S-\mathcal{S}$ which minimize the total expected cost per period for the stationary inventory system are solutions to the system of equations

$$m(\mathcal{S}, Q)=0 \tag{1}$$

and

$$\int_0^Q m(\mathcal{S}, x)\, dx=K \tag{2}$$

where $m(\mathcal{S}, x)$ satisfies the Volterra integral equation

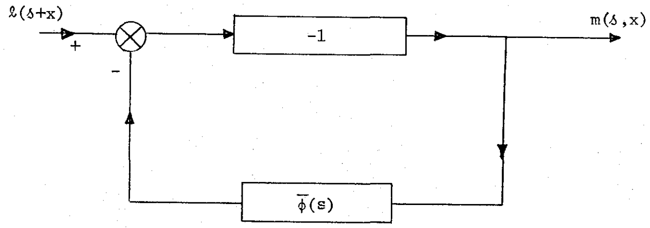


Figure 1—Feedback system simulating the equation

$$m(\mathcal{S}, x) = -l(\mathcal{S} + x) + \int_0^x m(\mathcal{S}, x-u)\, \phi\,(u)\, du$$

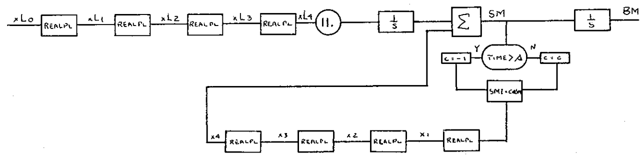


Figure 2—Flow chart for the digital simulation when h = 1, p = 10, n = 4 and a = .40

$$m(\mathcal{S},\, x) = -L'(\mathcal{S}+x) + \int_0^x m(\mathcal{S},\, x-\xi)\phi(\xi)\, d\xi \qquad (3)$$

Let

$$l(x) = L'(x)$$

$$\bar{l}(s,\, \mathcal{S}) = \mathcal{L}\{l(\mathcal{S}+x)\}$$

$$\bar{\phi}(s) = \mathcal{L}\{\phi(x)\}$$

$$\bar{m}(s,\, \mathcal{S}) = \mathcal{L}\{m(\mathcal{S},\, x)\}$$

Then, from taking the Laplace transforms on both sides of (3) and solving for $\bar{m}(\mathcal{S}, s)$ we obtain

$$\bar{m}(\mathcal{S},\, s) = \frac{-\bar{l}(\mathcal{S},\, s)}{1-\bar{\phi}(s)}$$

Thus, the integral equation (3) can be solved using the feedback system of Figure 1.

The optimal values $\mathcal{S}$ and Q can be determined by observing that the output function $m(\mathcal{S}, x)$ must satisfy equations (1) and (2). The interesting practical case which was investigated was the following:

Assume proportional holding and shortage costs, measured at end of period, then

$$L(x) = h\int_0^x (x-u)\phi(u)\, du + p\int_x^\infty (u-x)\phi(u)\, du \qquad (4)$$

In inventory terminology, h is the unit holding cost and p is the unit shortage cost. Differentiating (4) with respect to x, we obtain the following expression for $l(x)$:

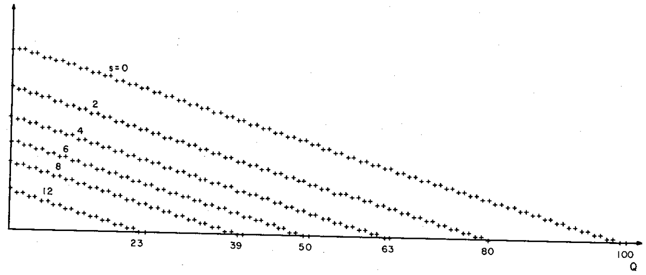


Figure 4—C.S.M.P. results for m(S, x) when n = 1, a = .1 and increments of .01

$$l(x) = (h+p)\int_0^x \phi(u)\, du - p \qquad (5)$$

We shall restrict our attention to the case when $\phi(\cdot)$ is a gamma density function of integer order n, i.e.,

$$\phi(\xi) = \frac{(a\xi)^{n-1}}{(n-1)!}ae^{-a\xi}a>0, \quad n=1, 2, \ldots \qquad (6)$$

Note then that in the s domain

$$\bar{\phi}(s) = a^n/(s+a)^n \quad n=1, 2, \ldots \qquad (7)$$

THE CONTINUOUS SYSTEM MODELING PROGRAM AND EXPERIMENTAL RESULTS

For illustrative purpose the numerical values of $h=1$, $p=10$ and $a=.10n$ were used. The selection of this particular value of a yields an expected demand of 10 units per period irrespective of the order of the gamma distribution. In Reference 5 a parallel connection was used to generate the gamma distribution. In contradistinction, a series connection is used in this paper involving a cascade of n filters of $a/(s+a)$. Thus, the

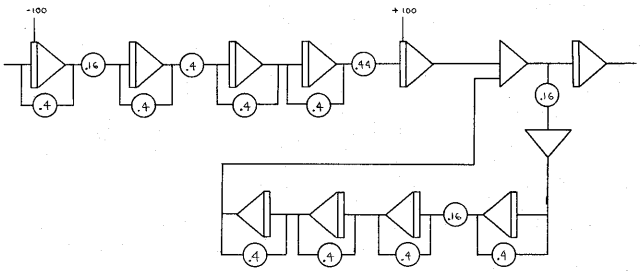


Figure 3—Analog equivalent of Figure 2

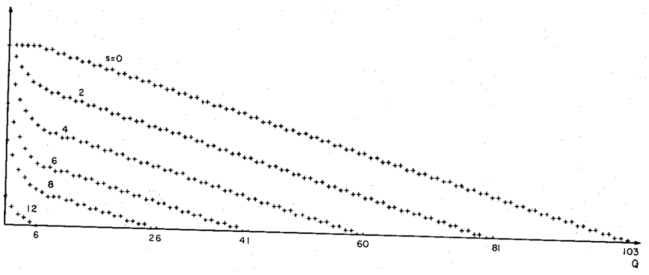


Figure 5—C.S.M.P. results for m(S, x) when n = 4, a = 4 and increments of .01

TABLE II—n = 1. Comparison Data for Alternative Methods Used in Solving for Q and K

	Q				K			
$\mathcal{S}$	Analog	Digital 1.0 (inc.)	Digital 0.1 (inc.)	Digital 0.01 (inc.)	Analog	Digital 1.0 (inc.)	Digital 0.1 (inc.)	Digital 0.01 (inc.)
0		100.0	100	100		500	499.9	499.4
2	78.9	81	79	80	315.7	332.6	319.2	320.1
4	62.9	64	63	63	200.6	211.0	202.3	203.1
6	49.7	51	50	50	124.9	132.0	126.3	126.8
8	38.9	40	39	39	76.4	81.0	77.4	77.7
12	22.8	23	23	23	26.2	28.0	26.6	26.8
Computing Cost						$ 2.45	$ 3.05	$ 8.52

entire problem was to be redesigned in such a way that the number of feedback loops as used in the original analog circuitry[5] was eliminated; instead, a serial first order loop approach was implemented. Figure 2 illustrates the flow chart of the digital simulation when $n=4$, while Figure 3 is the analog equivalent of Figure 2. Referring to Figure 2, when time is less than $\mathcal{S}$, $c=0$ and the feedback loop is idle. When time is greater than $\mathcal{S}$, then $c=-1$ thus initiating the feedback loop. At the same time, the last integrator marked $1/s$ is turned on on operate mode. The logic part has been left out in Figure 3 for the sake of simplicity and because it has well been defined in Reference 5. The C.S.M.P. program for $n=1$ to $n=19$ (Table I) was so designed as to print out the Q and K values when the function $m(\mathcal{S}, x)$ reaches zero.

TABLE III—n = 4. Comparison Data for Alternative Methods Used in Solving for Q and K

	Q				K			
$\mathcal{S}$	Analog	Digital 1.0 (inc.)	Digital 0.1 (inc.)	Digital 0.01 (inc.)	Analog	Digital 1.0 (inc.)	Digital 0.1 (inc.)	Digital 0.01 (inc.)
0	102.9	103	103	103	528.5	537.4	537.4	537
2	81.8	83	81	81	338.4	354.5	338.3	339.5
4	60.0	62	60	60	189.4	201.5	190.2	191.3
6	41.6	43	41	41	94.1	100.8	94	94.6
8	26.5	27	26	26	41.1	44.2	40.8	41.1
12	7.1	7	6	6	5.6	5.8	5.2	5.3
Computing Cost						$ 2.59	$ 3.66	$ 12.37

DISCUSSION

A significant marked improvement results for the output function $m(\mathcal{S}, x)$; this is illustrated in Figures 4, 5 and 6 for values of $n=1$, 4 and 19 respectively. The fact that the function $m(\mathcal{S}, x)$ can be generated for a gamma distribution of an order as high as $n=19$ is significant, particularly in the context of previous work.

Tables II and III report respectively some of the numerical results for $n=1$ and $n=4$ obtained from analog computation and digital computation using C.S.M.P. Increments of 1.00, .10 and .01 were used for comparison purposes.

It may be noticed that the order of magnitude of n depends on the size of the analog installation,[5] whereas the digital does not seem to have any limitations on the value of n at all. It is evident that the advantage of the analog system is not present in the discretized problem particularly when an immediate visual or graphical picture is desired for $m(\mathcal{S}, x)$.

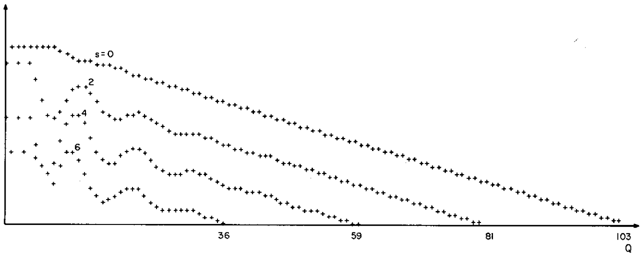


Figure 6—C.S.M.P. results for $m(\mathcal{S}, x)$ when $n = 19$, $a = 1.9$ and increments of .01

REFERENCES

1 K J ARROW T HARRIS Y MARSHAK
Optimal inventory policy
Econometrica 19 1951 pp 250-272

2 D M ROBERTS
Approximations to optimal policies in a dynamic inventory model
Studies in Applied Probability and Management Science—K J Arrow S Karlin H E Scarf eds Stanford University Press 1962 pp 207-229

3 B D SIVAZLIAN
Optimality and computation of the stationary (s, S) inventory control problem
Technical Report No 7 Project THEMIS Department of Industrial and Systems Engineering The University of Florida Gainesville Florida May 1968

4 _____
Dimensional and computational analysis in stationary (s, S) inventory problems with gamma distributed demand
Management Science Vol 17 1971 pp 307-311

5 _____ P E VALISALO
The utilization of a hybrid computer system in solving for optimal policy variables in inventory problems
Proceedings of the Sixth AICA Congress Munich Germany August 31-September 4 1970

6 A F VEINOTT JR H M WAGNER
Computing optimal (s, S) inventory policies
Management Science Volume 11 1965 pp 525-552

Bounds on multiprocessing anomalies and related packing algorithms

by R. L. GRAHAM

Bell Telephone Laboratories, Inc.
Murray Hill, New Jersey

INTRODUCTION

It has been known for some time that certain rather general models of multiprocessing systems frequently exhibit behavior which could be termed "anomalous," e.g., an *increase* in the number of processors of the system can cause an *increase* in the time used to complete a job.[42,36,20] In order to fully realize the potential benefits afforded by parallel processing, it becomes important to understand the underlying causes of this behavior and the extent to which the resulting system performance may be degraded.

In this paper we survey a number of theoretical results obtained during the past few years in connection with this topic. We also discuss many of the algorithms designed either to optimize or at least to improve the performance of the multiprocessor system under consideration.

The performance of a system or an algorithm can be measured in several rather different ways.[5] Two of the most common involve examining the *expected* behavior and the *worst-case* behavior of the object under consideration. Although knowledge of expected behavior is generally more useful in typical day-to-day applications, theoretical results in this direction require assumptions concerning the underlying probability distributions of the parameters involved and historically have been extremely resistant to attack.

On the other hand, there are many situations for which worst-case behavior is the appropriate measure (in addition to the fact that worst-case behavior does bound expected behavior). This type of analysis is currently a very active area of research, and theoretical insight into the worst-case behavior of a number of algorithms from various disciplines is now beginning to emerge (e.g., see References 7, 14, 15, 19, 20, 21, 26, 27, 29, 32, 33, 44, 47, and especially Reference 41.) It is this latter measure of performance which will be used on the models and algorithms of this paper. Since it is essential to have on hand the worst examples one can think of before conjecturing and (hopefully) proving bounds on worst-case behavior, numerous such examples will be given throughout the text.

Before concluding this section, it seems appropriate to make a few remarks concerning the general area of these topics. Recent years have seen the emergence of a vital and important new discipline, often called "analysis of algorithms." As its broad objective, it seeks to obtain a deeper understanding of the nature of algorithms. These investigations range, for example, from the detailed analysis of the behavior of a specific sorting routine, on one hand, to the recent negative solution[†] to Hilbert's Tenth Problem by Matijasevič[37] and Julia Robinson,[43] on the other. It is within this general framework that the present discussion should be viewed.

A GENERAL MULTIPROCESSING SYSTEM

Let us suppose we have n (abstract) identical processors P_i, $i=1, \ldots, n$, and we are given a set of tasks $\mathcal{T}=\{T_1, \ldots, T_r\}$ which is to be processed by the P_i. We are also given a partial order[††] $<$ on $\mathcal{T}$ and a function $\mu:\mathcal{T}\rightarrow(0, \infty)$. Once a processor P_i begins to execute a task T_j, it works without interruption until the completion of that task,[†††] requiring altogether $\mu(T_j)$ units of time. It is also required that the partial order be respected in the following sense: If $T_i<T_j$ then T_j cannot be started until T_i has been completed. Finally, we are given a sequence $L=(T_{i_1}, \ldots, T_{i_r})$, called the

[†] Which roughly speaking shows that there is no universal algorithm for deciding whether a diophantine equation has solutions or not.

[††] See Reference 31 for terminology.

[†††] This is known as *nonpreemptive* scheduling as opposed to *preemptive* scheduling in which the execution of a task may be interrupted.[5]

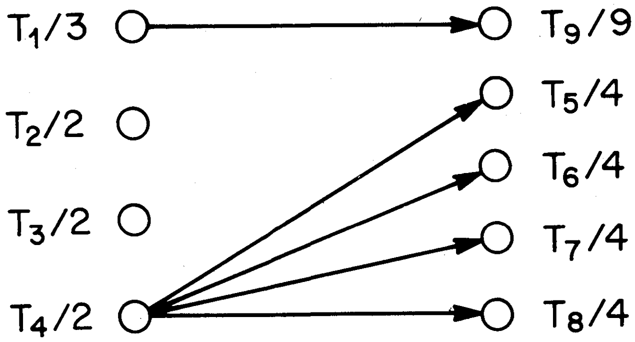


Figure 1—A simple graph G

(*priority*) *list* (or schedule) consisting of some permutation of all the tasks $\mathcal{T}$. The P_i execute the T_j as follows: Initially, at time 0, all the processors (instantaneously) scan the list L from the beginning, searching for tasks T_i which are "ready" to be executed, i.e., which have no predecessors under $<$. The first ready task T_j in L which P_i finds immediately begins to be executed by P_i; P_i continues to execute T_j for the $\mu(T_j)$ units of time required to complete T_j. In general, at any time a processor P_i completes a task, it immediately scans L for the first available ready task to execute. If there are currently no such tasks, then P_i becomes idle. We shall also say in this case that P_i is executing an *empty task* which we denote by φ (or φ_i). P_i remains idle until some other P_k completes a task, at which time P_i (and, of course, P_k) immediately scans L for ready tasks which may now exist because of the completion of T_j. If two (or more) processors both attempt to start executing a task, it will be our convention to assign the task to the processor with the smaller index. The least time at which all tasks of T have been completed will be denoted by ω.

We consider an example which illustrates the working of the preceding multiprocessing system and various anomalies associated with it. We indicate the partial order $<$ on T and the function μ by a *directed graph*[†] $G(<, \mu)$. In $G(<, \mu)$ the vertices correspond to the T_i and a directed edge from T_i to T_j denotes $T_i < T_j$. The vertex T_j of $G(<, \mu)$ will usually be labeled with the symbols $T_j/\mu(T_j)$. The activity of each P_i is conveniently represented by a *timing diagram D* (also known as a Gantt chart.[2] D consists of n horizontal half-lines (labeled by the P_i) in which each line is a time axis starting from time 0 and is subdivided into segments labeled according to the corresponding activity of the P_i. In Figure 1 we show a simple graph G.

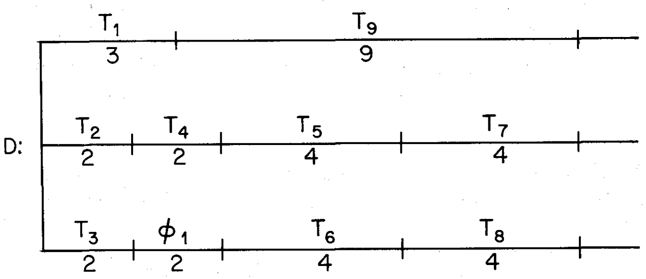


Figure 2—The timing diagram D for G

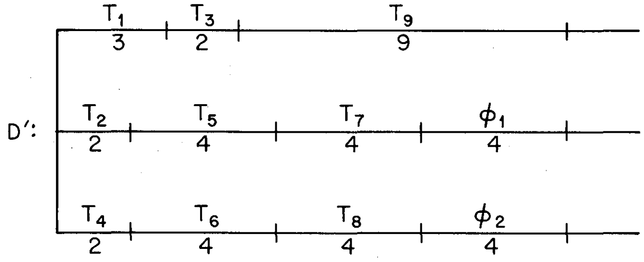


Figure 3—The timing diagram D′ when L′ is used

In Figure 2, we give the corresponding timing diagram D assuming that the list $L = (T_1, T_2, \ldots, T_9)$ is used with three processors. The finishing time is $\omega = 12$.

Note that on D we have labeled the intervals above by the task and below by the length of time needed to execute the task.

It is evident from the definition of ω that it is a function of L, μ, $<$ and n. Let us vary each of these four parameters in the example and see the resulting effect this variation has on ω.

(i) Replace L by $L' = (T_1, T_2, T_4, T_5, T_6, T_3, T_9, T_7, T_8)$, leaving μ, $<$ and n unchanged (Figure 3). For the new list L', $\omega' = \omega'(L', \mu, <, n) = 14$.

(ii) Change $<$ to $<'$ by removing $T_4 < T_5$ and $T_4 < T_6$.

For the new partial order $<'$, $\omega' = \omega'(L, \mu, <', n) = 16$ (Figure 4).

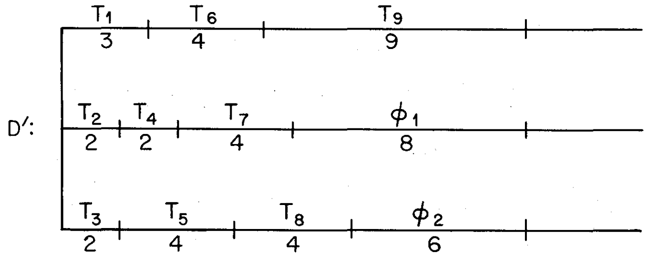


Figure 4—The timing diagram D′ when <′ is used

[†] For terminology in graph theory, see Reference 23.

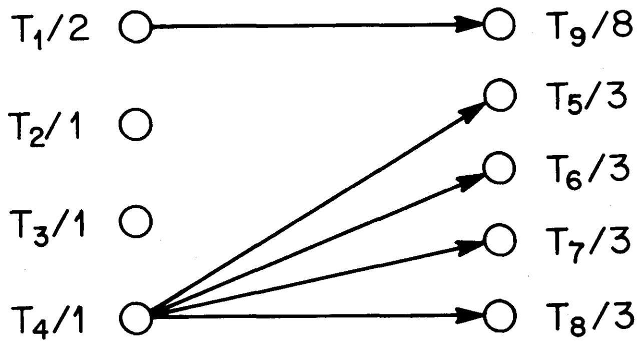


Figure 5—The new graph G using μ'

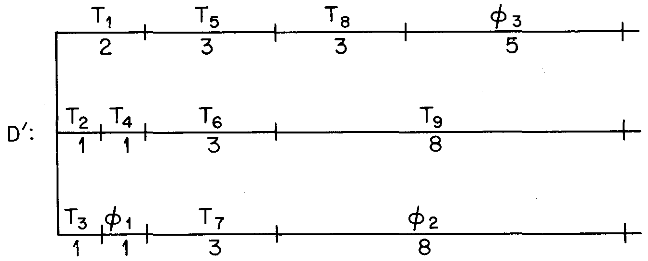


Figure 6—The timing diagram D′ when μ' is used

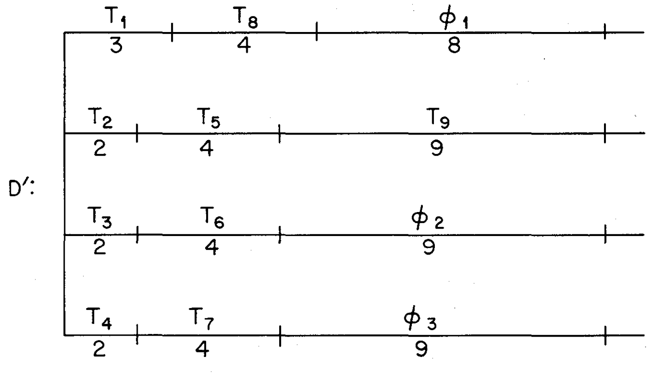


Figure 7—The timing diagram D′ when 4 processors are used

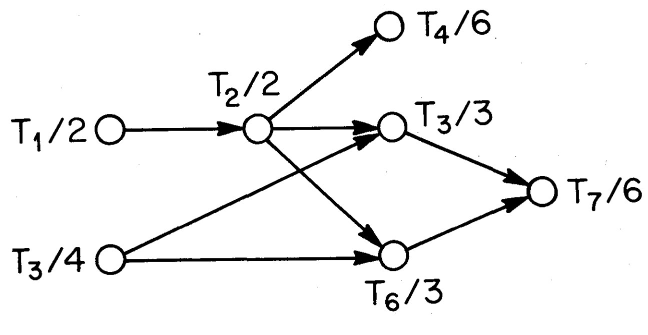


Figure 8—Another simple graph

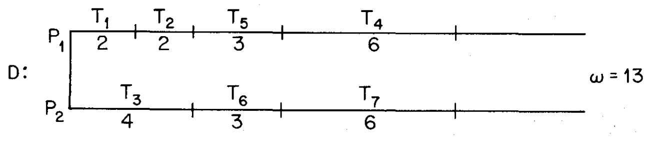


Figure 9—The timing diagram D using the list L

(iii) Decrease μ to μ' by defining $\mu'(T_i) = \mu(T_i) - 1$ for all i. In this case $G(<, \mu)$ is shown in Figure 5.

The corresponding timing diagram D' is shown in Figure 6 where we see $\omega' = \omega'(L, \mu', <, n) = 13$.

(iv) Increase n from three to four (Figure 7). In this case $\omega' = 15$!

Note that in (iii) by using the list $L'' = (T_1, T_2, T_3, T_4, T_9, T_5, T_6, T_7, T_8)$ we can reduce the finishing time to 10 which is less than the 12 of the original unshortened problem. This does not always have to occur though, as the example given in Figure 8 shows.

When the (optimal) list $L = (T_1, T_2, T_3, T_5, T_6, T_4, T_7)$ is used, $\omega = 13$ (Figure 9).

If all execution times are decreased by one unit then *all* lists generate the same finishing time $\omega' = 14$. A typical D' is shown in Figure 10.

A GENERAL BOUND

The examples in the preceding section show that contrary to what might generally be expected, *relaxing* $<$, *decreasing* μ or *increasing* n can all cause ω to *increase*. It is natural to inquire into the extent to which these changes can affect the finishing time ω. The right measure turns out to be the ratio of the possible finishing times, and a bound is given in the following result.

Theorem.[19,20]

Suppose we are given a set of tasks $\mathcal{T}$, which we wish to execute twice. The first time we use a time function μ, a partial order $<$, a list L and a multiprocessing system composed of n processors. The second time we

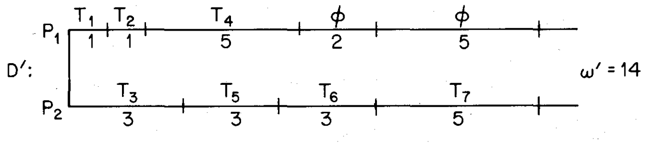


Figure 10—A timing diagram D using shortened times

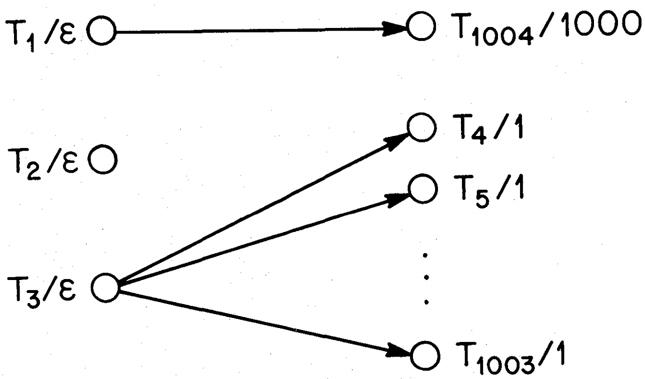


Figure 11—A graph G

use a time function $\mu' \le \mu$, a partial order[†] $<' \subseteq <$, a list L' and a multiprocessing system composed of n' processors. Let ω and ω' denote the respective finishing times. Then

$$\omega'/\omega \le 1+(n-)1/n'. \tag{1}$$

Furthermore, this bound is best possible in the sense that the right-hand side cannot be replaced by any smaller function of n and n'.

For $n = n'$, the bound becomes

$$\omega'/\omega \le 2-1/n \tag{2}$$

In fact, examples[††] can be given[19] which show that the bound in (2) can be achieved (to within an arbitrary $\epsilon > 0$) by varying any *one* of the three parameters L, μ and $<$. Note that (2) implies that using the worst possible list instead of the best possible list still only results in an increase in ω of at most a factor of $2-1/n$.

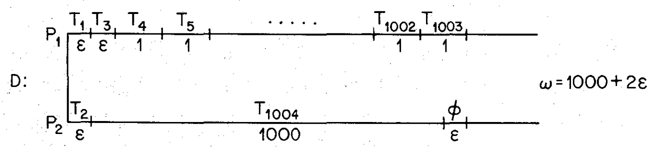


Figure 12—2 processors are used to execute the tasks of G

When $n = 1$, (1) implies $\omega'/\omega \le 1$ which agrees with the obvious fact that the aforementioned changes in L, μ, $<$ and n can never cause increase in the running time when compared to that for single processor. On the other hand, when $n > 1$ then even a large increase in the number of processors (but with a fixed list L) can cause ω to increase as the example given in Figure 11 shows (where $0 < \epsilon < 1$).

If the tasks of G are executed by two processors using the list $L = (T_1, T_2, T_3, \ldots, T_{1004})$ then $\omega = 1000 + 2\epsilon$ (Figure 12).

If the tasks of G are executed by 1000 processors using the same list L then $\omega' = 1001 + \epsilon$ (Figure 13). However, it is always true that the use of a suitable list will prevent ω from increasing because of an increase in n (e.g., in this case, take $L = (T_{1004}, T_1, T_2, \ldots, T_{1003})$).

We mention here that even if inserted idleness and preemption[†] are allowed in the first run, it can be shown that ω/ω' is still bounded above by $2-1/n$.[3,18]

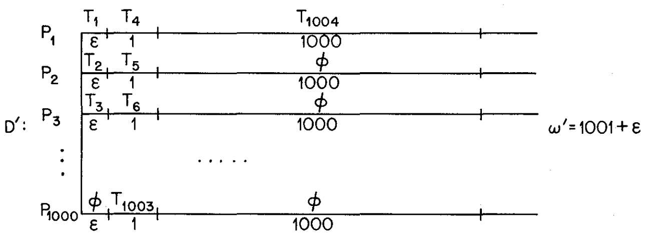


Figure 13—1000 processors are used to execute the tasks of G

SOME SPECIAL BOUNDS

We next consider results arising from attempts at finding lists which keep the ratio of ω/ω_0 close to one. Since for any given problem, there are just finitely many possible lists then one might be tempted to say: "Examine them all and select the optimum." Not much insight is needed, however, to realize that due to the explosive growth of functions such as $n!$, this is not a realistic solution. What one is looking for instead is an algorithm which will guarantee that ω/ω_0 is reasonably close to one provided we are willing to expend an appropriate amount of energy applying the algorithm. Unfortunately, no general results of this type presently exist. There is a special case, however, in which steps in this direction have been taken. This is the case in which $<$ is empty, i.e., there are no precedence constraints between the tasks. We shall restrict ourselves to this case for the remainder of this section.

Suppose, for some (arbitrary) k, the k tasks with the largest values of μ have been selected[†] and somehow

[†] Since a partial order on $\mathfrak{T}$ is a subset of $\mathfrak{T} \times \mathfrak{T}$, $<' \subseteq <$ has the obvious meaning.

[††] Recent examples of M. T. Kaufman (personal communication) show that in many cases the bounds of (1) and (2) can be achieved even when $\mu(T) = 1$ for all T.

[†] i.e., holding a processor idle when it could be busy and interrupting the execution of a task before completion.

[†] Recent results of Blum, Floyd, Pratt, Rivest and Tarjan[41] allow this to be done in no more than $6r$ binary comparisons where r denotes the number of tasks.

arranged in a list L_k which is optimal with respect to this set of k tasks (i.e., for no other list can this set of k tasks finish earlier). Form the list $L^{(k)}$ by adjoining the remaining tasks arbitrarily but so that they follow all the tasks of L_k. Let $\omega(k)$ denote the finishing time using this list with a system of n processors. If ω_0 denotes the global optimum, i.e., the minimum possible finishing time over all possible lists then the following result holds.

Theorem.[20]

$$\frac{\omega(k)}{\omega_0} \leq 1+\frac{1-1/n}{1+[k/n]} \tag{3}$$

For $k \equiv 0 (\text{mod } n)$ this bound is best possible.

For the case of $k \equiv 0 (\text{mod } n)$ the following example establishes the optimality of the bound from below.

For $1 \leq i \leq k+1+n(n-1)$, define $\mu(T_i)$ by

$$\mu(T_i)=\begin{cases} n & \text{for } 1 \leq i \leq k+1, \\ 1 & \text{for } k+2 \leq i \leq k+1+n(n-1). \end{cases}$$

For this set of tasks and the list $L(k)=(T_1, \ldots, T_k, T_{k+2}, \ldots, T_{k+1+n(n-1)}, T_{k+1})$ we have $\omega(k)=k+2n-1$.

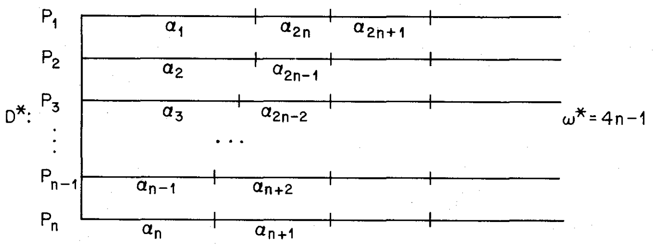


Figure 14—The timing diagram D* using the decreasing list L*

Since $\omega_0=k+n$, and $k \equiv 0 (\text{mod } n)$ then

$$\frac{\omega(k)}{\omega_0}=1+\frac{1-1/n}{1+[k/n]}$$

as asserted.

For $k=0$, (3) reduces to (2) while for $k=n$ we have

$$\omega(n)/\omega_0 \leq 3/2-1/2n. \tag{4}$$

The required optimal assignment of the largest n tasks to the n processors is immediate—just assign each of these tasks to a different processor. For $k=2n$, (3) reduces to

$$\omega(2n)/\omega_0 \leq 4/3-1/3n, \tag{5}$$

a bound which will soon be encountered again.

An important property of (3) is that the right-hand side tends to 1 as k gets larger compared to n. Thus, in order for $\omega(k)$ to be assured of being within 10 percent of the minimum value ω_0, for example, it suffices to optimally schedule the largest $9n$ tasks.

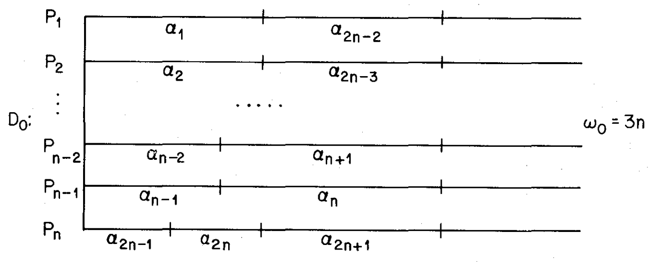


Figure 15—The timing diagram D₀ using an optimal list

Another heuristic technique for approximating the optimal finishing time ω_0 is to use the "decreasing" list† $L^*=(T_{i_1}, T_{i_2}, \ldots)$ where $\mu(T_{i_1}) \geq \mu(T_{i_2}) \geq \ldots$. The corresponding finishing time ω^* satisfies the following inequality.

Theorem.[20]

$$\omega^*/\omega_0 \leq 4/3-1/3n. \tag{6}$$

This bound is best possible.

For $n=2$, (6) yields a bound of 7/6 which is exactly the ratio obtained from the canonical example with five tasks having execution times of 3, 3, 2, 2 and 2. More generally, the following example shows that (6) is exact. $\mathfrak{T}$ consists of $r=2n+1$ independent tasks T_k with $\mu(T_k)=\alpha_k=2n-[(k+1)/2]$ for $1 \leq k \leq 2n$ and $\mu(T_{2n+1})=\alpha_{2n+1}=n$ (where $[x]$ denotes the greatest integer not exceeding x). Thus

$$(\alpha_1, \alpha_2, \ldots, \alpha_{2n+1})$$
$$=(2n-1, 2n-1, 2n-2, 2n-2, \ldots, n+1, n+1, n, n, n)$$

In Figure 14, $\mathfrak{T}$ is executed using the decreasing list $L^*=(T_1, T_2, \ldots, T_{2n+1})$.

In Figure 15, $\mathfrak{T}$ is executed using an optimal list.

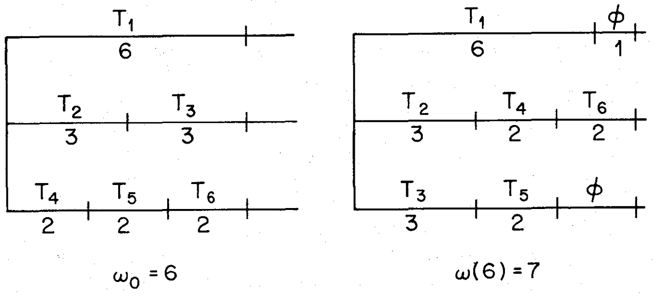


Figure 16—Example illustrating difference between L* and L(2n)

† Such a list can be formed in essentially no more than $r \log r/\log 2$ binary comparisons.[33]

It is interesting to observe that although the right-hand sides of (5) and (6) are the same, arranging the tasks by decreasing execution time does not necessarily guarantee that the largest $2n$ tasks will be executed optimally by L^*. An example showing this (due to J. H. Spencer (personal communication)) is given in Figure 16. The execution times of the tasks are (6, 3, 3, 2, 2, 2) and three processors are used.

The following result shows that if none of the execution times is large compared to the sum of all the execution times then ω^* cannot be too far from ω_0.

Theorem.[18]

If $<$ is empty and

$$\max \mu(T) / \sum_{T} \mu(T) \leq \beta$$

then

$$\omega^*/\omega_0 \leq 1+n\beta. \tag{7}$$

Another approach is to start with a timing diagram resulting from some arbitrary list and then make *pairwise interchanges* between tasks executed by pairs of processors which decrease the finishing time, until this can no longer be done. If ω' denotes the final finishing time resulting from this operation then it can be shown[18] that

$$\omega'/\omega_0 \leq 2-2/(n+1) \tag{8}$$

and, furthermore, this bound cannot be improved.

SOME ALGORITHMS FOR OPTIMAL LISTS

There seems little doubt that even for the case when $<$ is empty, $\mu(T)$ is an integer, and $n=2$, any algorithm which determines an optimal list for any set of tasks must be essentially enumerative† in its computational efficiency. This problem can be rephrased as follows:

Given a sequence $S=(s_1, \ldots, s_r)$ of positive integers find a choice of $\epsilon_i = \pm 1$ such that

$$\left| \sum_{k=1}^{r} \epsilon_k s_k \right|$$

is minimized.

Thus any hope for efficient algorithms which produce optimal schedules for more general multiprocessing problems seems remote indeed. There are several special cases, however, for which such algorithms exist.

For the case when $\mu(T)=1$ for all tasks T and $<$ is a forest,† Hu[28] has shown that the following algorithm generates an optimal list L_0.

(i) Define the level $\lambda(T)$ of any terminal†† task T to be 1.
(ii) If T' is an immediate successor of T, define $\lambda(T)$ to be $\lambda(T')+1$.
(iii) Form the list $L_0=(T_{i_1}, T_{i_2}, \ldots, T_{i_r})$ in order of decreasing λ values, i.e., $\lambda(T_{i_1}) \geq \lambda(T_{i_2}) \geq \cdots \geq \lambda(T_{i_r})$.

Theorem.[28]

L_0 is an optimal list when $\mu(T)=1$ for all T and $<$ is a tree.

The only other case for which an efficient algorithm is currently known is when $\mu(T)=1$ for all T, $n=2$ and $<$ is arbitrary. In fact, two quite distinct algorithms have been given. One of these, due to Fujii, Kasami and Ninomiya[11,12] is based on a matching algorithm for bipartite graphs of Edmonds[8] and appears to be of order $O(r^3)$. The other, due to Coffman and Graham,[3] uses the following labeling technique:

Assuming there are r tasks, each task T will be assigned a unique label $\alpha(T) \epsilon \{1, 2, \ldots, r\}$.

(i) Choose an arbitrary terminal task T_0 and define $\alpha(T_0)=1$.
(ii) Assume the values $1, 2, \ldots, k-1$ have been assigned for some $k \leq r$. For each unlabeled task T having all its immediate successors already labeled, form the decreasing sequence $M(T)=(m_1, m_2, \ldots, m_5)$ of the labels of T's immediate successors. These $M(T)$ are lexicographically† ordered. Choose a *minimal* $M(T')$ in this order and define $\alpha(T')=k$.
(iii) Form the list $L^*=(T_{i_1}, T_{i_2}, \ldots, T_{i_r})$ according to decreasing α values, i.e., $\alpha(T_{i_1}) > \alpha(T_{i_2}) > \cdots > \alpha(T_{i_r})$.

Theorem.[3]

L^* is an optimal list when $\mu(T)=1$ for all T and $n=2$.

† More precisely, the number of steps it may require cannot be bounded by a fixed polynomial in the number of bits of information needed to specify the input data.

† This means that every task T has at most one immediate successor T′, i.e., T < T′ and for no T″ is T < T″ < T′. Actually, by adding a dummy task T_0 preceded by all other tasks, < can be made into a *tree*, without loss of generality.

†† i.e., a task with no successor.

† i.e., dictionary order, so that (5, 4, 3) precedes (6, 2) and (5, 4, 3, 2).

This algorithm has been shown to be of order $O(r^2)$ and so, in a sense, is best possible since the partial order $<$ can also have this order of number of elements.

The increase in complexity of this algorithm over Hu's algorithm seems to be due to the greatly increased structure an *arbitrary* partial order may have when compared to that of a tree. Even relatively simple partial orders can defeat many other algorithms which might be thought to be optimal for this case. The example in Figure 17 illustrates this.

An optimal list for this example is $L^* = (T_{10}, T_9, \ldots, T_1)$ where we assume $\mu(T_i) = 1$ for all i. Any algorithm which allows T_{10} not to be executed first is not optimal, as, for example, executing tasks on the basis of the longest chain to a terminal task (i.e., according to levels), or executing tasks on the basis of the largest number of successors.

For $n > 2$ and $\mu(T) = 1$ for all T, the algorithm no longer produces optimal lists as the example in Figure 18 shows.

It would be interesting to know the worst-case behavior of the lists produced by this algorithm for general n.

The current state of affairs here seems to be similar to that of the job-shop scheduling problem[5,30] for which optimal schedules can be efficiently generated when $n = 2$, while for $n > 2$ no satisfactory algorithms are known.

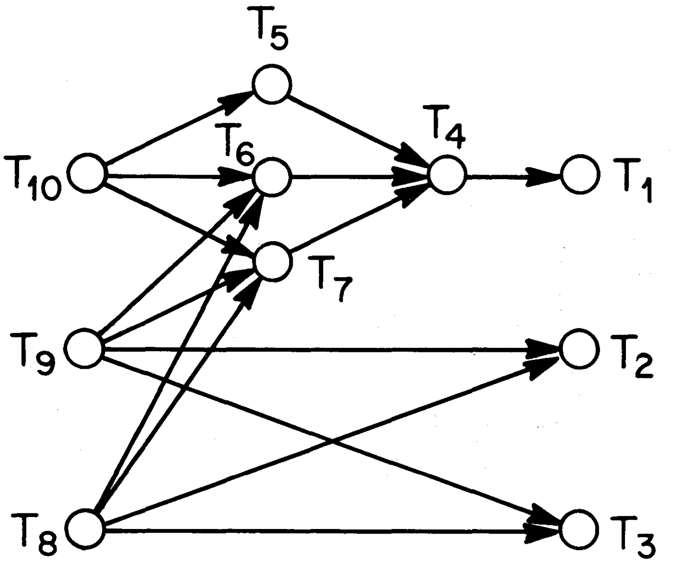


Figure 17—A useful graph for counterexamples

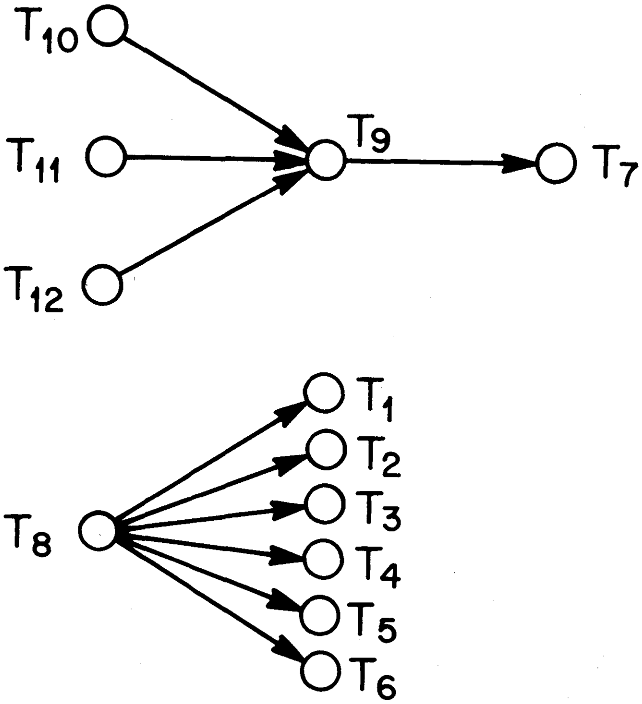


Figure 18—A counterexample to optimality when $n = 3$

A DUAL PROBLEM

Up to this point, we have generally regarded the number of processors as fixed and asked for the list L which (approximately) minimizes the finishing time ω. We now invert this question as follows: For a fixed deadline ω^*, we ask for a list L which when used will (approximately) minimize the number of processors needed to execute all tasks by the time ω^*. Of course, the two questions are essentially equivalent, and so no efficient algorithms are known for the general case.† Nevertheless, a number of recent results are now available for several special cases and these will be the subject of this section.

We first make a few remarks concerning the general case. It is not hard to see that if λ^* denotes the length of the longest chain†† in the partially-ordered set of tasks $\mathfrak{J}$, then we must have $\omega^* \geq \lambda^*$. Otherwise, no number of processors could execute all tasks of $\mathfrak{J}$ in ω^* time units. On the other hand, if sufficiently many processors are available then all tasks of $\mathfrak{J}$ can be executed in time λ^*. In fact, if m^* denotes the maximal number of mutually *incomparable*† tasks of $\mathfrak{J}$, then it is never necessary to have more than m^* processors in the system since clearly no more than m^* can be in operation at any one time.

† Both of these questions are raised in Reference 10.

†† i.e., a sequence $T_{i_1} < T_{i_2} < \cdots < T_{i_m}$ with $\sum_k \mu(T_{i_k})$ maximal.

† T_i and T_j are incomparable if neither $T_i < T_j$ nor $T_j < T_i$ hold.

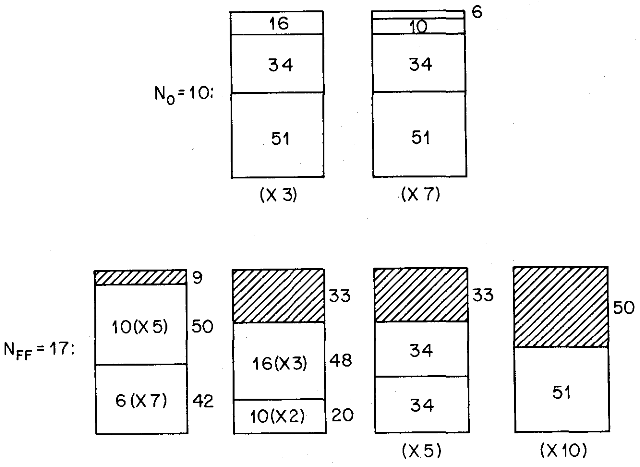


Figure 19—An example with N_{FF}/N_0 = 17/10

For the case in which $\mu(T)=1$ for all tasks T, lower bounds for both problems are provided by results of Hu.[28] In this case, if $\lambda(T)$ denotes the *level* of a task T as defined in the preceding section, let m denote the *maximum* level of any task in $\mathfrak{J}$ and for $0 \leq k \leq m$, let $\Lambda(k)$ denote the number of tasks having level strictly greater than k.

Theorem.[28]

If n processors can execute $\mathfrak{J}$ with finishing time ω then

$$\omega \geq \max_{0 \leq k \leq m} (k+\Lambda(k)/n). \qquad (9)$$

For other early results dealing with these and related problems, the reader may consult References 1, 5, 13, 24, 38, 42, 45, and 46.

For the remainder of this section, we restrict ourselves to the special case in which there are no precedence constraints on the tasks. In this case the second problem becomes a special case of the one-dimensional cutting stock problem[15,17] as well as a special case of the assembly-line balancing problem.[5]

We can also think of the problem in the following terms. We are given a set of objects ${}_i$O with O_i having weight $\alpha_i=\mu(T_i)$, $1 \leq i \leq r$. We have at our disposal an unlimited supply of boxes B_j, each with a maximum capacity of ω^* units of weight. It is required to assign all the objects to the minimum number N_0 of boxes subject to the constraint that the total weight assigned to any box can be no more than ω^*. In this form, this question takes the form of a typical loading or packing problem.[9] Integer linear programming algorithms have been given[9,16,17,25] for obtaining optimal solutions for this problem, but the amount of computation necessary soon becomes excessive as the size of the problem grows.

Several heuristic algorithms have been suggested[9,15,40] for approximating N_0. One of these, which we call the "first-fit" algorithm, is defined as follows: For a given list $L=(\alpha_{i_2}, \alpha_{i_2}, \ldots, \alpha_{i_r})$, the α_{i_k} are successively assigned in order of increasing k, each to the box B_j of lowest index into which it can validly be placed. The number of boxes thus required will be denoted by $N_{FF}(L)$, or just N_{FF}, when the dependence on L is suppressed.

If L is chosen so that $\alpha_{i_1} \geq \alpha_{i_2} \geq \cdots \geq \alpha_{i_r}$ then the first-fit algorithm using this list is called the "first-fit decreasing" algorithm and the corresponding $N_{FF}(L)$ is denoted by N_{FFD}.

Instead of first-fit, one might instead assign the next α_k in a list L to the box in which the resulting unused capacity is minimal. This is called the "best-fit" algorithm and N_{BF} will denote the number of boxes required in this case. The corresponding definitions of "best-fit decreasing" and N_{BFD} are analogous to first-fit decreasing and N_{FFD} and are omitted.

One of the first questions which arises concerning these algorithms is the extent by which they can ever deviate from N_0. Only for N_{FF} is the behavior accurately known.

Theorem.[47,15]

For any $\epsilon > 0$, if N_0 is sufficiently large then

$$N_{FF}/N_0 < 17/10+\epsilon. \qquad (10)$$

The 17/10 in (10) is best possible. An example for which $N_{FF}/N_0=17/10$ is given in Figure 19 (where

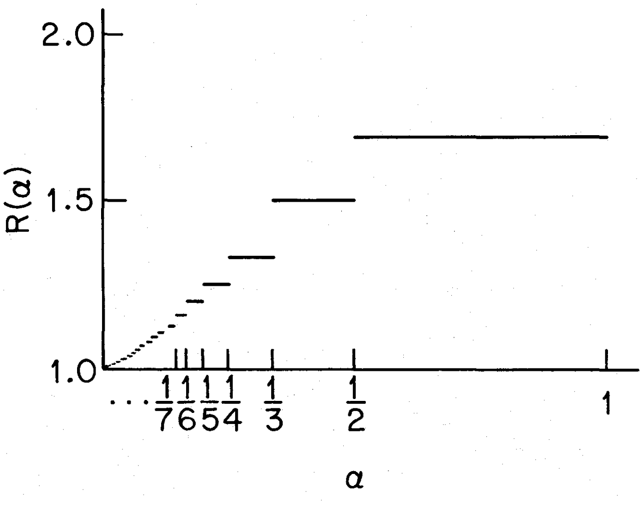


Figure 20—The function R(α)

$\omega^* = 101$). The multiplicity of types of boxes and objects are given in parentheses.

For any $\epsilon > 0$ examples can be given with $N_{FF}/N_0 \geq 17/10 - \epsilon$ and N_0 arbitrarily large. It appears, however, that for N_0 sufficiently large, N_{FF}/N_0 is strictly less than 17/10.

In order to achieve a ratio N_{FF}/N_0 close to 17/10 it is necessary[15] to have some of the α_i exceed $\omega^*/2$. Conversely, if all α_i are small compared to ω^* then N_{FF}/N_0 must be relatively close to one. This is stated precisely in the following result.

Theorem.[15]

Suppose $\max \alpha_i/\omega^* \leq \alpha$. Then for any $\epsilon > 0$, if N_0 is sufficiently large then

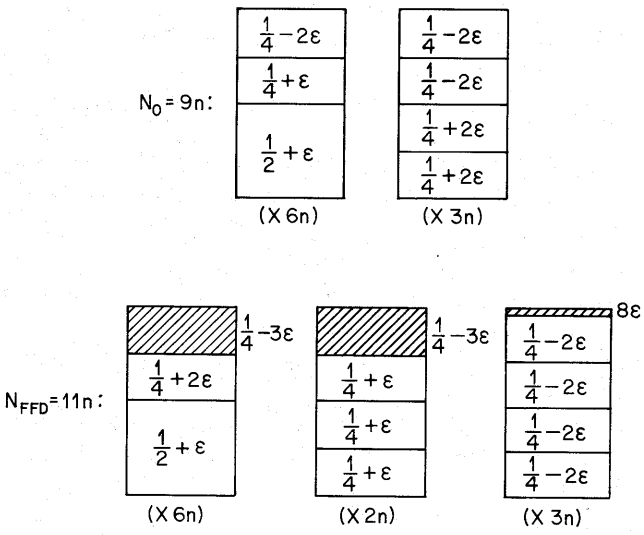
$$N_{FF}/N_0 - \epsilon \leq \begin{cases} 17/10 \text{ for } \alpha > \frac{1}{2}, \\ 1 + \lfloor \alpha^{-1} \rfloor^{-1} \text{ for } 0 < \alpha \leq \frac{1}{2}. \end{cases} \qquad (11)$$

The right-hand side of (11) cannot be replaced by any smaller function of α.

We denote the right-hand side of (11) by $R(\alpha)$ and illustrate it in Figure 20.

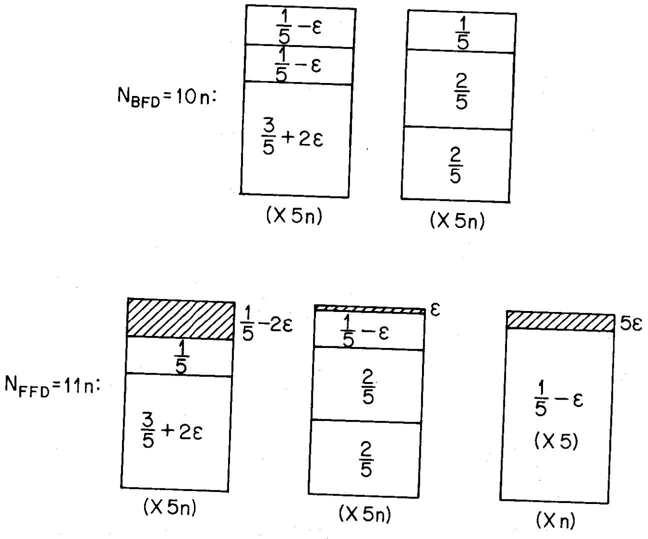
It is conjectured[15] that the worst-case behavior of N_{BF}/N_0 is the same as that of N_{FF}/N_0 but this has not yet been established. It is known that $R(\alpha)$ is also a lower bound for N_{BF}/N_0 when $\max \alpha_i \leq \alpha\omega^*$.

As one might suspect, N_{FFD}/N_0 cannot differ from 1 by as much as N_{FF}/N_0 can. This is shown in the following result.



$N_0 = 9n$:
$\frac{1}{4} - 2\varepsilon$
$\frac{1}{4} + \varepsilon$
$\frac{1}{2} + \varepsilon$
(X 6n)
$\frac{1}{4} - 2\varepsilon$
$\frac{1}{4} - 2\varepsilon$
$\frac{1}{4} + 2\varepsilon$
$\frac{1}{4} + 2\varepsilon$
(X 3n)
$N_{FFD} = 11n$:
$\frac{1}{4} - 3\varepsilon$
$\frac{1}{4} + 2\varepsilon$
$\frac{1}{2} + \varepsilon$
(X 6n)
$\frac{1}{4} - 3\varepsilon$
$\frac{1}{4} + \varepsilon$
$\frac{1}{4} + \varepsilon$
$\frac{1}{4} + \varepsilon$
(X 2n)
8ε
$\frac{1}{4} - 2\varepsilon$
$\frac{1}{4} - 2\varepsilon$
$\frac{1}{4} - 2\varepsilon$
$\frac{1}{4} - 2\varepsilon$
(X 3n)

Figure 21—An example with $N_{FFD}/N_0 = 11/9$ and N_0 large



$N_{BFD} = 10n$:
$\frac{1}{5} - \varepsilon$
$\frac{1}{5} - \varepsilon$
$\frac{3}{5} + 2\varepsilon$
(X 5n)
$\frac{1}{5}$
$\frac{2}{5}$
$\frac{2}{5}$
(X 5n)
$N_{FFD} = 11n$:
$\frac{1}{5} - 2\varepsilon$
$\frac{1}{5}$
$\frac{3}{5} + 2\varepsilon$
(X 5n)
ε
$\frac{1}{5} - \varepsilon$
$\frac{2}{5}$
$\frac{2}{5}$
(X 5n)
5ε
$\frac{1}{5} - \varepsilon$
(X 5)
(X n)

Figure 22—An example with $N_{FFD}/N_{BFD} = 11/10$ and N_0 large

Theorem.[15]

For any $\epsilon > 0$, if N_0 is sufficiently large then

$$N_{FFD}/N_0 < 5/4 + \epsilon. \qquad (12)$$

The example in Figure 21 shows that

$$N_{FFD}/N_0 \geq 11/9 \qquad (13)$$

is possible for N_0 arbitrarily large. It is conjectured that the 5/4 in (12) can be replaced by 11/9; this has been established[15] for some restricted classes of α_i.

The preceding remarks also apply, *mutatis mutandis*, to the ratio N_{BFD}/N_0. This is implied by the following somewhat surprising result.

Theorem.[15]

If $\max \alpha_i/\omega^* \geq 1/5$ then $N_{FFD} = N_{BFD}$.

The quantity 1/5 above cannot be replaced by any smaller number as the example in Figure 22 shows.

This example raises the whole question concerning the extent by which the numbers N_{FF}, N_{BF}, N_{FFD} and N_{BFD} may differ among themselves (assuming that N_0 is large). The example in Figure 22 shows that $N_{FFD}/N_{BFD} \geq 11/10$ is possible for arbitrarily large N_0. On the other hand, the example of Figure 23 shows that $N_{BFD}/N_{FFD} \geq 13/12$ is possible for arbitrarily large N_0. These two examples represent the worst behavior of N_{FFD}/N_{BFD} and N_{BFD}/N_{FFD} currently known.

Another algorithm which has been proposed[40] proceeds by first selecting from all the α_i a subset which packs B_1 as well as possible, then selecting from the

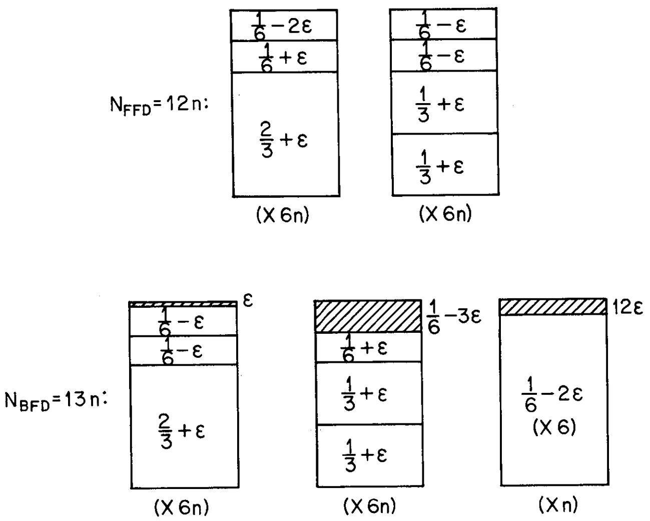

$N_{FFD}=12n$:
$\frac{1}{6}-2\varepsilon$
$\frac{1}{6}+\varepsilon$
$\frac{2}{3}+\varepsilon$
(X 6n)
$\frac{1}{6}-\varepsilon$
$\frac{1}{6}-\varepsilon$
$\frac{1}{3}+\varepsilon$
$\frac{1}{3}+\varepsilon$
(X 6n)
$N_{BFD}=13n$:
ε
$\frac{1}{6}-\varepsilon$
$\frac{1}{6}-\varepsilon$
$\frac{2}{3}+\varepsilon$
(X 6n)
$\frac{1}{6}-3\varepsilon$
$\frac{1}{6}+\varepsilon$
$\frac{1}{3}+\varepsilon$
$\frac{1}{3}+\varepsilon$
(X 6n)
12ε
$\frac{1}{6}-2\varepsilon$
(X 6)
(X n)

Figure 23—An example with $N_{BFD}/N_{FFD} = 13/12$ and N_0 large

remaining α_i a subset which packs B_2 as well as possible, etc. Although more computation would usually be required for this algorithm than for the first-fit decreasing algorithm it might be hoped that the number $\bar{N}$ of boxes required is reasonably close to N_0. This does *not* have to be the case, however, since examples exist for any $\epsilon > 0$ for which N_0 is arbitrarily large and

$$\bar{N}/N_0 > \sum_{n=1}^{\infty} 1/(2^n-1) - \epsilon. \qquad (14)$$

The quantity

$$\sum_{n=1}^{\infty} 1/(2^n-1) = 1.606695\ldots$$

in (14) is conjectured[15] to be best possible.

Some of the difficulty in proving many of the preceding results and conjectures seems to stem from the fact that a *decrease* in the values of the α_i may result in an *increase* in the number of boxes required. For example, if the weights (760, 395, 395, 379, 379, 241, 200, 105, 105, 40) are packed into boxes of capacity 1000 using the first-fit decreasing algorithm then we find $N_{FFD}=3$ which is optimal. However, if all the weights are decreased by one, so that now the weights (759, 394, 394, 378, 378, 240, 199, 104, 104, 39) are packed into boxes of capacity 1000 using the first-fit decreasing algorithm, we have $N_{FFD}=4$ which is clearly not optimal. In fact, the following example shows that N_{FF} can *increase* when some of the α_i are *deleted.* In Figure 24 the list $L = (7, 9, 7, 1, 6, 2, 4, 3)$ is used with the first-fit algorithm to pack boxes of capacity 13, resulting in $N_{FF}(L) = 3$.

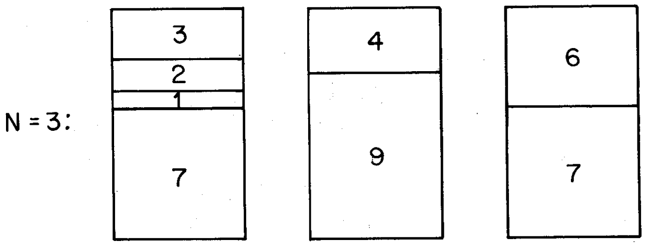


Figure 24—An optimal packing using L

If the number 1 is deleted from L, to form the list $L' = (7, 9, 7, 6, 2, 4, 3)$, then we see in Figure 25 that $N_{FF}(L') = 4$.

DYNAMIC TASK SELECTION

As an alternative to having a fixed list L prior to execution time which determines the order in which tasks should be attempted, one might employ an algorithm which determines the scheduling of the tasks in a dynamic way, making decisions dependent on the intermediate results of the execution. Unfortunately, no efficient algorithms of this type are known which can prevent the worst possible worst-case behavior.

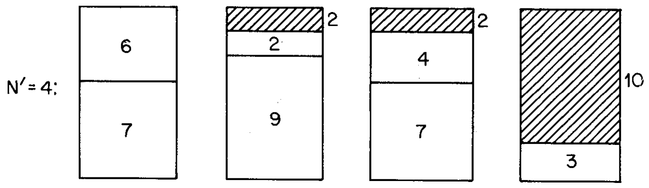


Figure 25—A nonoptimal packing using the deleted list L′

Perhaps the most natural candidate is the "critical-path" algorithm,[5] which always selects as the next task to be executed, that available task which belongs to the longest[†] chain of currently unexecuted tasks. This type of analysis forms the basis for many of the project planning techniques which have been developed such as PERT, CPM, etc.[39] Its worst-case behavior can be bad as possible as the example in Figure 26 shows (where $0 < \epsilon < 1$).

If ω_{CP} denotes the finishing time for three processors when the critical path algorithm is used on the example in Figure 26, we have $\omega_{CP} = 2n - 1 - 2\epsilon$. However, the

[†] Where, as mentioned before, the length of a chain $T_{i1} < \cdots < T_{im}$ is $\sum_k \mu(T_{ik})$.

optimal solution has $\omega_0 = n$, giving a ratio of

$$\omega_{CP}/\omega_0 = 2-1+2\epsilon/n \tag{15}$$

Since ϵ may be chosen arbitrarily close to 0, then ω_{CP}/ω_0 may be arbitrarily close to the previous bound of $2-1/n$.

This example also applies to the algorithm which selects as the next task to be executed, that available task for which the sum of the execution times of all its successors is maximal.

It may be true that the critical path algorithm may not have such extreme worst-case behavior when all the $\mu(T_i)$ are nearly equal, although not too much in this direction can be hoped for as the example in Figure 27 shows.
In this example, where n processors are used and all $\mu(T_i)=1$, $\omega_{CP}=2n$ is possible depending on how some of the ties are broken. Since $\omega_0 = n+1$ then we obtain a ratio

$$\omega_{CP}/\omega_0 = 2-2/(n+1) \tag{16}$$

which may be the maximum value possible in this case.

CONCLUDING REMARKS

As the reader will have gathered from the preceding discussion, there are certainly more questions than answers available at this point in time. We take this opportunity to comment on several of these questions, indicating what seem to the author to be fruitful directions for further research.

1. What efficient algorithms exist for preventing worst-case behavior in the general multiprocessor problem from approaching the $2-1/n$ bound? It certainly seems that just as in the case when there are no precedence constraints between tasks, it should be possible to show in some quantitative sense, that if one is willing to use more complex algorithms, one can be guaranteed of getting closer to the optimum.

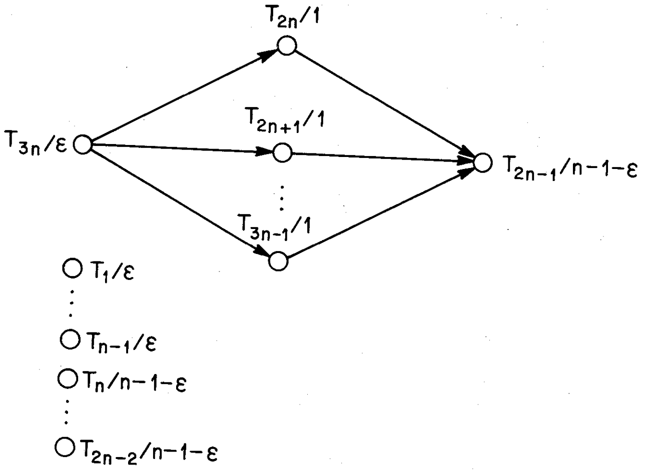


Figure 26—Example which causes worst possible critical path algorithm behavior

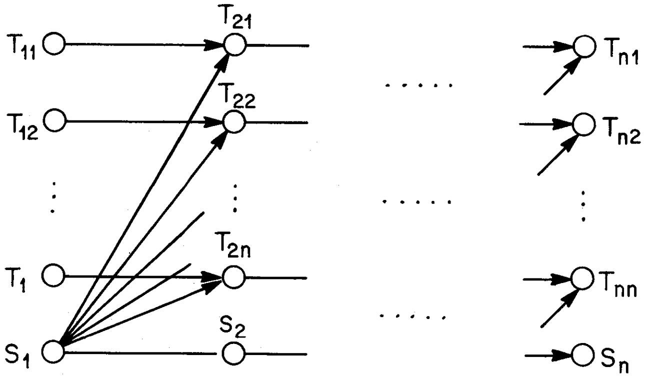


Figure 27—Example which can cause bad critical path algorithm behavior

2. There seems to be little possibility that an efficient† algorithm exists for the determination of optimal schedules for the general multiprocessor problem. Recent work of S. Cook[6] and R. M. Karp (personal communication) helps to clarify some of these issues. They show that a large class of combinatorial problems (one of which is a special case of this problem) are equivalent in this respect, i.e., either they all have efficient algorithms or none do. However, up to now everyone has been singularly unsuccessful in proving the nonexistence of such algorithms. The time seems ripe to remedy this unsatisfactory situation.

3. In the other direction, it seems likely that efficient algorithms should exist for other special cases. For example, good candidates would appear to be the cases $n=3$, $\mu(T)=1$ for all T and $n=2$, $\mu(T)=1$ or 2 for all T.

4. In reference to the dual (cutting stock) problem of an earlier section, a number of interesting open questions remain, in addition to those already mentioned. For example, one could allow boxes of different capacities and study the behavior of $N_{FF}(L)/N_{FF}(L')$ for a fixed set of weights, as a function of the lists L and L', the ordering of the boxes, the distribution of the capacities and weights, etc.[9] It would also be of interest to examine two-dimensional analogues of these problems in view of the applicability of the results (e.g., see References 16 and 17).

5. Much of the motivation for studying worst-case behavior is derived from the possible insight the results

† In the sense of Edmonds.[8]

may provide for the typical or expected behavior of the system. Very little has been rigorously established in this direction so far although some empirical results are available. For example, in simulation studies involving fairly large task sets, from two to nine processors, and unit task execution times, Manacher[36] reports that in roughly four-fifths of his runs, optimal lists fail to remain optimal when the task execution times are (randomly) slightly perturbed. In other studies, Krone[34] has investigated the typical behavior of several algorithms applied to tasks with no precedence constraints. In particular, he compared the finishing times ω^* and ω' obtained by using the "decreasing" list L^* and by using stabilized pairwise interchanges, respectively. He found that usually $\omega' < \omega^*$ when execution times had a large variance (in spite of the fact that their worst-case behavior is reversed). On the other hand, when the execution times were more nearly equal, ω^* was very good and, in fact, frequently optimal.

In view of the remarkable progress which has occurred in this and other branches of computer science during the past decade, there is little doubt in my mind that the answers to these and many other related questions will be uncovered in the not-too-distant future.

REFERENCES

1 R BELLMAN
Mathematical aspects of scheduling theory
SIAM Jour of App Math 4 1956 pp 168-205

2 W CLARK
The Gantt chart
3rd ed Pitman and Sons London 1952

3 E G COFFMAN JR R L GRAHAM
Optimal scheduling for two-processor systems
To appear Acta Informatica 2 1972

4 E G COFFMAN JR P J DENNING
Operating systems theory
To appear Prentice-Hall 1972

5 R W CONWAY W L MAXWELL L W MILLER
Theory of scheduling
Addison-Wesley Reading 1967

6 S COOK
The complexity of theorem proving
Proc 3rd Annual ACM Symp on Theory of Computing 1971 pp 151-158

7 W L EASTMAN S EVEN I M ISAACS
Bounds for the optimal scheduling of n jobs on m processors
Manag Sci 11 No 2 1964 pp 268-279

8 J EDMONDS
Paths, trees and flowers
Can Jour of Math 17 1965 pp 449-467

9 S EILON N CHRISTOFIDES
The loading problem
Manag Sci 17 No 5 1971 pp 259-268

10 L R FORD JR D R FULKERSON
Flows in networks
Princeton Univ Press Princeton 1962

11 M FUJII T KASAMI K NINOMIYA
Optimal sequencing of two equivalent processors
SIAM Jour of App Math 17 No 3 1969 pp 784-789

12 ____
Erratum
SIAM Jour of App Math 20 No 1 1971 p 141

13 D R FULKERSON
Scheduling in project networks
Proc IBM Scientific Computing Symp on Combinatorial Problems IBM Corp New York 1966 pp 73-92

14 M R GAREY R L GRAHAM
Asymptotic performance bounds on the splitting algorithm for binary testing
To appear in Acta Informatica

15 M R GAREY R L GRAHAM J D ULLMAN
An analysis of some cutting stock algorithms
To appear

16 P C GILMORE R E GOMORY
A linear programming approach to the cutting stock problem
Oper Res 9 1961 pp 849-859

17 ____
A linear programming approach to the cutting stock problem II
Oper Res 11 1963 pp 863-888

18 R L GRAHAM
Unpublished

19 ____
Bounds for certain multiprocessing anomalies
Bell Sys Tech Jour 45 No 9 1966 pp 1563-1581

20 ____
Bounds on multiprocessing timing anomalies
SIAM Jour of App Math 17 No 2 1969 pp 416-429

21 ____
On sorting by comparisons
Computers in Number Theory Ed by A O L Atkin and B J Birch Acad Press New York 1971 pp 263-269

22 A L GUTJAHR G L NEMHAUSER
An algorithm for the line balancing problem
Manag Sci 11 No 2 1964 pp 308-315

23 F HARARY
Graph theory
Addison-Wesley Reading 1969

24 M HELD R M KARP
A dynamic programming approach to sequencing problems
SIAM Jour of App Math 10 No 2 1962

25 M HELD R M KARP R SHARESKIAN
Assembly-line balancing dynamic programming with precedence constraints
Oper Res 11 1963 pp 442-459

26 J HOPCROFT R TARJAN
Planarity testing in V log V steps
Information Processing 1971 Proc of IFIP Congress 71 1971

27 J HOPCROFT R M KARP
An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs
IEEE Conf Record of the Twelfth Annual Symp on Switching and Automata 1971

28 T C HU
Parallel sequencing and assembly line problems
Oper Res 9 No 6 1961 pp 841-848

29 F K HWANG S LIN
Optimal merging of 2 elements with n elements
Acta Informatica 1 1971 pp 145-158

30 S M JOHNSON
Optimal two- and three-stage production schedules with setup times included
Nav Res Log Quart 1 No 1 1954. Also Industrial Scheduling ed by F F Muth and G L Thompson Prentice-Hall Englewood Cliffs NJ 1963

31 J L KELLEY
General topology
Van Nostrand Princeton 1955

32 V KLEE G J MINTY
How good is the simplex algorithm?
Inequalities III ed by O Shisha Acad Press New York 1972 pp 159-175

33 D E KNUTH
The art of computer programming
Vol 3 Addison-Wesley 1972

34 M J KRONE
Heuristic programming applied to scheduling problems
PhD dissertation EE Dept Princeton Univ 1970

35 E L LAWLER
On scheduling problems with deferral costs
Manag Sci 11 No 2 1964 pp 280-288

36 G K MANACHER
Production and stabilization of real-time task schedules
JACM 14 No 3 1967 pp 439-465

37 Ju v MATIJASEVIC
Enumerable sets are diophantine
(In Russian) Doklady 191 1970 pp 279-282

38 R McNAUGHTON
Scheduling with deadlines and loss functions
Manag Sci 6 No 1 1959 pp 1-12

39 J J MODER C R PHILLIPS
Project management with CPM and PERT Reinhold New York 1964

40 R C PRIM
Personal communication

41 E M REINGOLD
Establishing lower bounds on algorithms: a survey
AFIPS Conf Proc 40 1972

42 P RICHARDS
Parallel programming
Report TD-B60-37 Technical Operations Inc 1960

43 J ROBINSON
Diophantine decision problems
Studies in Number Theory ed by W J LeVeque MAA Studies in Math Vol 6 1969

44 J M ROBSON
An estimate of the store size necessary for dynamic storage allocation
JACM 18 No 3 1971 pp 416-423

45 J G ROOT
Scheduling with deadlines and loss functions on k parallel machines
Manag Sci 11 No 3 1965 pp 460-475

46 M H ROTHKOPF
Scheduling independent tasks on parallel processors
Manag Sci 12 No 5 1966 pp 437-447

47 J D ULLMAN
The performance of a memory allocation algorithm
Tech Report No 100 EE Dept Princeton Univ 1971

Computation of recursive programs—Theory vs practice

by ZOHAR MANNA

Stanford University
Stanford, California

This note is actually an informal exposition of a part of a recent paper by Manna, Ness and Vuillemin.[1] We have two main purposes in this note. First, we present some known results about computation of recursive programs, emphasizing some differences between the theoretical and practical approaches. Second, we introduce the computational induction method for proving properties of recursive programs. It turns out that most known methods for proving properties of programs are very closely related to the computational induction method. We illustrate this point by showing how Floyd's inductive assertions method for proving properties of "flowchart programs" can be expressed in terms of computational induction on recursive programs.

The reader should be aware that some of the results presented in this note hold only under certain restrictions which are ignored in this informal presentation.

RECURSIVE PROGRAMS

To simplify our discussion, we shall restrict ourselves to a particularly simple language, chosen because of its similarity to familiar languages such as ALGOL or LISP. A program in our language, called a *recursive program,* is of the form

$$F(\bar{x}) \Leftarrow \tau[F](\bar{x}),$$

where $\tau[F](\bar{x})$ is a composition of known functions and the function variable F, applied to the individual variables $\bar{x}=(x_1, x_2, \ldots, x_n)$. The following, for example, is a recursive program over the integers

$$P_0\colon\quad F(x_1, x_2) \Leftarrow \textit{if } x_1=x_2 \textit{ then } x_2+1 \textit{ else } F(x_1, F(x_1-1, x_2+1)).$$

We allow our known functions to be partial, i.e., they may be *undefined* for some arguments. This is quite natural, since our known functions represent the result of some computation, and a computation process may in general give results for some arguments and run indefinitely for others. We include as limiting cases of partial functions, the partial functions defined for all arguments (called total functions) as well as the partial function undefined for all arguments.

Let us consider now the following partial functions:

$f_1(x_1, x_2)$: x_1+1

$f_2(x_1, x_2)$: *if* $x_1 \geq x_2$ *then* x_1+1 *else* x_2-1, and

$f_3(x_1, x_2)$: *if* $(x_1 \geq x_2) \wedge (x_1-x_2 \text{ even})$ *then* x_1+1
else undefined.

These three functions have an interesting common property: For each i $(1 \leq i \leq 3)$, when we replace all occurrences of F in the program P_0 by f_i, the lefthand side and the righthand side of the $\Leftarrow$ yield identical partial functions, namely,
$f_i(x_1, x_2) \equiv \textit{if } x_1=x_2 \textit{ then } x_2+1 \textit{ else } f_i(x_1, f_i(x_1-1, x_2+1))$.
It is straightforward to see that the equality holds for $f_1(x_1, x_2)$, for example, since in this case we get

$$x_1+1 \equiv \textit{if } x_1=x_2 \textit{ then } x_2+1 \textit{ else } x_1+1,$$

which is clearly true. One can similarly verify that f_2 and f_3 have the same property. We therefore say that the functions f_1, f_2 and f_3 are *fixpoints* of the recursive program P_0.

Among the three functions, f_3 has one important special property: for any (x_1, x_2) such that $f_3(x_1, x_2)$ is defined, i.e., $(x_1 \geq x_2) \wedge (x_1-x_2 \text{ even})$, both $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$ are also defined and have the same value as $f_3(x_1, x_2)$. For short, we say that f_3 is "less defined" than f_1 and f_2 and denote this by $f_3 \subseteq f_1$ and $f_3 \subseteq f_2$. It can be shown that f_3 has this property not only with respect to f_1 and f_2 but with respect to all fixpoints of the recursive program P_0. Moreover, $f_3(x_1, x_2)$ is the *only* function having this property; f_3 is therefore said to be the *least (defined) fixpoint of* P_0.

One of the most important results related to this topic is due to Kleene,[2] who showed that *every recursive program P has a unique least fixpoint (denoted by f_P).*

In discussing our recursive programs, the key problem is: *What is the partial function f defined by a recursive program P?* There are two viewpoints.

(a) *Fixpoint approach*: Let it be the unique least fixpoint f_P.
(b) *Computational approach*: Let it be the computed function f_C for some given computation rule C (such as "call by name" or "call by value").

Now we come to a very interesting point. All the theory of proving properties of recursive programs is actually based on the assumption that the function defined by a recursive program is exactly the least fixpoint f_P. That is, the fixpoint approach is adopted. Unfortunately, many commonly used programming languages imply computation rules for evaluating such recursive programs (such as "call by value!!") which do not necessarily lead to the least fixpoint.

Let us consider, for example, the following recursive program over the integers

$$P_1: \quad F(x_1, x_2) \Leftarrow \mathit{if}\ x_1=0\ \mathit{then}\ 1\ \mathit{else}\ F(x_1-1, F(x_1-x_2, x_2)).$$

The least fixpoint f_{P_1} can be shown to be

$$f_{P_1}(x_1, x_2): \quad \mathit{if}\ x_1 \geq 0\ \mathit{then}\ 1\ \mathit{else\ undefined}.$$

However, the computed function f_C, where C is "call by value," turns out to be

$$f_C(x_1, x_2): \quad \mathit{if}\ x_1=0 \vee [x_1>0 \wedge x_2>0 \wedge (x_2 \text{ divides } x_1)]\ \mathit{then}\ 1\ \mathit{else\ undefined}.$$

Thus, f_C is properly less defined than f_{P_1}, e.g., $f_C(1, 0)$ is undefined while $f_{P_1}(1, 0)=1$.*

There are two ways to view this problem: either (a) theoreticians are wasting their time by developing methods for proving properties of programs which "do not exist" in practice. They should concentrate their efforts in developing direct methods for proving properties of programs as they are actually executed; or (b) existing computer systems should be modified, since they are computing recursive programs in a way which prevents their user from benefiting from the results of the theory of computation. Language designers and implementors should look for efficient computation rules which always lead to the least fixpoint. "Call by name," for example, is such a computation rule, but unfortunately it often leads to very inefficient computations. An efficient computation rule which always leads to the least fixpoint can be obtained by modifying "call by value" so that the evaluation of the arguments of the function variable F are delayed as long as possible.[4]

One way to cope with the problem would be to develop translation techniques, so that for every given recursive program P and computation rule C, we can construct a recursive program P' such that the computed function of P is identical to the least fixpoint of P'. The verification techniques can then be applied to P'. This is probably the way many verification systems are going to work in the future. The main reason for adopting this approach is the existence of a very powerful method, the computational induction method, for proving properties of the least fixpoint of recursive programs. Most known methods for proving properties of programs can be expressed in terms of the computational induction method, as illustrated later. The computational induction method has two important advantages over the other methods: First, it is very convenient for machine implementation;[5] and second, termination and equivalence proofs can be handled in exactly the same way as correctness proofs.

THE COMPUTATIONAL INDUCTION METHOD

We now describe the *computational induction* method for proving properties of recursive programs. The idea is essentially to prove properties of the least fixpoint f_P of a given recursive program P by induction on the level of recursion.

Let us consider, for example, the recursive program

$$P_2: \quad F(x) \Leftarrow \mathit{if}\ x=0\ \mathit{then}\ 1\ \mathit{else}\ x \cdot F(x-1),$$

over the natural numbers. The least fixpoint function $f_{P_2}(x)$ of this recursive program is the factorial function $x!$.

Let us denote now by $f^i(x)$ the partial function indicating the "information" we have after the ith level of recursion. That is,

$f^0(x)$ is *undefined* (for all x);

$f^1(x)$ is $\mathit{if}\ x=0\ \mathit{then}\ 1\ \mathit{else}\ x \cdot f^0(x-1)$, i.e.,

$$\mathit{if}\ x=0\ \mathit{then}\ 1\ \mathit{else\ undefined};$$

$f^2(x)$ is $\mathit{if}\ x=0\ \mathit{then}\ 1\ \mathit{else}\ x \cdot f^1(x-1)$, i.e.,

$\mathit{if}\ x=0\ \mathit{then}\ 1\ \mathit{else}\ x \cdot (\text{if } x-1=0\ \mathit{then}\ 1\ \mathit{else\ undefined})$, or in short,

$\mathit{if}\ x=0 \vee x=1\ \mathit{then}\ 1\ \mathit{else\ undefined};$

etc.

* It can actually be shown, in general, that for every recursive program P and any computation rule C, if f_C and f_P are not identical, then f_C must be less defined than f_P (Cadiou [3]).

In general, for every i, $i\geq 1$,

$$f^i(x) \text{ is } \textit{if } x=0 \textit{ then } 1 \textit{ else } x\cdot f^{i-1}(x-1),$$

which is

$$\textit{if } x<i \textit{ then } x! \textit{ else undefined.}$$

This sequence of functions has a limit which is exactly the least fixpoint of the recursive program; that is,

$$\lim_i f^i(x) \equiv x!$$

The important point is that this is actually the case for any recursive program P. That is, if P is a recursive program of the form

$$F(\bar{x}) \Leftarrow \tau[F](\bar{x}),$$

and $f^i(\bar{x})$ is defined by

$$f^0(\bar{x}) \text{ is } \textit{undefined} \text{ (for all } \bar{x}),$$

and

$$f^i(\bar{x}) \text{ is } \tau[f^{i-1}](\bar{x}) \text{ for } i\geq 1,^*$$

then

$$\lim_i f^i(\bar{x}) \equiv f_P(\bar{x}).$$

This suggests an induction rule for proving properties of f_P. To show that some property φ holds for f_P, i.e., $\varphi(f_P)$, we show that $\varphi(f^i)$ holds for all $i\geq 0$, and therefore we may conclude that $\varphi(\lim_i f^i)$, i.e., $\varphi(f_P)$, holds.

There are two ways to state this rule. Both are essentially equally powerful. These are actually the rules for simple and complete induction on the level of recursion.

(a) *simple induction*
if $\varphi(f^0)$ *holds and* $\forall i[\varphi(f^i)\Rightarrow\varphi(f^{i+1})]$ *holds, then* $\varphi(f_P)$ *holds.*

(b) *complete induction*
if $\forall i\{[(\forall j \textit{ s.t. } j<i)\varphi(f^j)]\Rightarrow\varphi(f^i)\}$ *holds,*** *then* $\varphi(f_P)$ *holds.*

The simple induction rule is essentially the "μ-rule" suggested by deBakker and Scott,[6] while the complete induction rule is the "truncation induction rule" of Morris.[7]

Example: Consider the two recursive programs[7]

$$P_3:\quad F(x,y)\Leftarrow \textit{if } p(x) \textit{ then } y \textit{ else } h(F(k(x),y)),$$

and

$$P_4:\quad G(x,y)\Leftarrow \textit{if } p(x) \textit{ then } y \textit{ else } G(k(x),h(y)).$$

For our purpose there is no need to specify the domain of the programs or the meaning of p, h and k. We would like to prove, using the two forms of induction, that

$$f_{P_3}(x,y)\equiv g_{P_4}(x,y) \text{ for all } x \text{ and } y.$$

Proof by simple induction

If we restrict ourselves to simple induction, it is much more convenient to prove a stronger result than the desired one. This often simplifies proofs by induction by allowing a stronger induction assumption, even though we have to prove a stronger result. So, we actually show that

$$\varphi(f_{P_3},g_{P_4}):\quad \forall x\forall y\{[f_{P_3}(x,y)\equiv g_{P_4}(x,y)] \wedge [g_{P_4}(x,h(y))\equiv h(g_{P_4}(x,y))]\}$$

holds. We proceed in two steps:

(a) $\varphi(f^0,g^0)$, i.e., $\forall x\forall y\{[f^0(x,y)\equiv g^0(x,y)] \wedge [g^0(x,h(y))\equiv h(g^0(x,y))]\}$.

$$\forall x\forall y\{[\textit{undefined}\equiv\textit{undefined}] \wedge [\textit{undefined}\equiv\textit{undefined}]\}.$$

(b) $\forall i[\varphi(f^i,g^i)\Rightarrow\varphi(f^{i+1},g^{i+1})]$.

We assume

$$\forall x\forall y\{[f^i(x,y)\equiv g^i(x,y)] \wedge [g^i(x,h(y))\equiv h(g^i(x,y))]\},$$

and prove

$$\forall x\forall y\{[f^{i+1}(x,y)\equiv g^{i+1}(x,y)] \wedge [g^{i+1}(x,h(y))\equiv h(g^{i+1}(x,y))]\}.$$

$$\begin{aligned} f^{i+1}(x,y) &\equiv \textit{if } p(x) \textit{ then } y \textit{ else } h(f^i(k(x),y))\\ &\equiv \textit{if } p(x) \textit{ then } y \textit{ else } h(g^i(k(x),y))\\ &\equiv \textit{if } p(x) \textit{ then } y \textit{ else } g^i(k(x),h(y))\\ &\equiv g^{i+1}(x,y).\end{aligned}$$

$$\begin{aligned} g^{i+1}(x,h(y)) &\equiv \textit{if } p(x) \textit{ then } h(y) \textit{ else } g^i(k(x),h^2(y))\\ &\equiv \textit{if } p(x) \textit{ then } h(y) \textit{ else } h(g^i(k(x),h(y)))\\ &\equiv h(\textit{if } p(x) \textit{ then } y \textit{ else } g^i(k(x),h(y)))\\ &\equiv h(g^{i+1}(x,y)).\end{aligned}$$

Proof by complete induction

Using complete induction we can prove the desired result directly; that is, we prove that

$$\varphi(f_{P_3},g_{P_4}):\quad \forall x\forall y[f_{P_3}(x,y)\equiv g_{P_4}(x,y)]$$

* Where J[f^{i-1}] is the result of replacing f^{i-1} for all occurrences of F in J[F]

** Note that this indicates implicitly the need to prove $\varphi(f^0)$ separately, since for $i = 0$ there is no j s.t. $j < i$.

holds. This is done by showing first that $\varphi(f^0, g^0)$ and $\varphi(f^1, g^1)$ hold, and then that $\varphi(f^i, g^i)$ holds for all $i \geq 2$. (We treat the cases for $i=0$ and $i=1$ separately, since to prove $\varphi(f^i, g^i)$ we use the induction hypothesis for both $i-1$ and $i-2$.)

(a) $\varphi(f^0, g^0)$, i.e., $\forall x \forall y [f^0(x, y) \equiv g^0(x, y)]$.

$$\forall x \forall y [\mathit{undefined} \equiv \mathit{undefined}].$$

(b) $\varphi(f^1, g^1)$, i.e., $\forall x \forall y [f^1(x, y) \equiv g^1(x, y)]$.

$$\begin{aligned} f^1(x, y) &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else}\ h(f^0(k(x), y)) \\ &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else\ undefined} \\ &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else}\ g^0(k(x), h(y)) \\ &\equiv g^1(x, y). \end{aligned}$$

(c) $(\forall i \geq 2)[\varphi(f^{i-2}, g^{i-2}) \wedge \varphi(f^{i-1}, g^{i-1}) \Rightarrow \varphi(f^i, g^i)]$
We assume

$$\forall x \forall y [f^{i-2}(x, y) \equiv g^{i-2}(x, y)]$$

and

$$\forall x \forall y [f^{i-1}(x, y) \equiv g^{i-1}(x, y)],$$

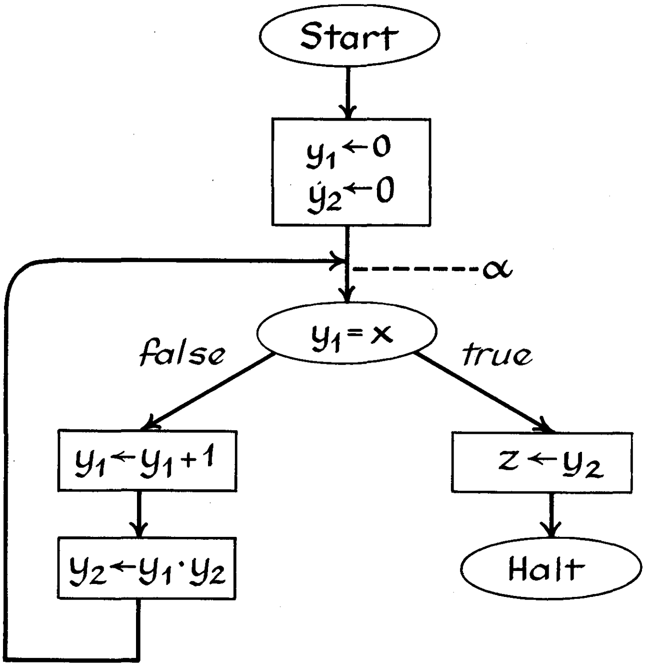
and deduce

$$\forall x \forall y [f^i(x, y) \equiv g^i(x, y)].$$

$$\begin{aligned} f^i(x, y) &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else}\ h(f^{i-1}(k(x), y)) \\ &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else}\ h(g^{i-1}(k(x), y)) \\ &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else}\ h(\mathit{if}\ p(k(x))\ \mathit{then}\ y \\ &\qquad\qquad \mathit{else}\ g^{i-2}(k^2(x), h(y))) \\ &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else}\ (\mathit{if}\ p(k(x))\ \mathit{then}\ h(y) \\ &\qquad\qquad \mathit{else}\ h(g^{i-2}(k^2(x), h(y)))) \\ &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else}\ (\mathit{if}\ p(k(x))\ \mathit{then}\ h(y) \\ &\qquad\qquad \mathit{else}\ h(f^{i-2}(k^2(x), h(y)))) \\ &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else}\ f^{i-1}(k(x), h(y)) \\ &\equiv \mathit{if}\ p(x)\ \mathit{then}\ y\ \mathit{else}\ g^{i-1}(k(x), h(y)) \\ &\equiv g^i(x, y). \end{aligned}$$

THE INDUCTIVE ASSERTIONS METHOD

The most widely used method for proving properties of "flowchart programs" is presently the *inductive assertions method*, suggested by Floyd[8] and Naur.[9] It can be shown that for any proof by inductive assertions, there is a naturally corresponding computational induction proof. We shall illustrate the inductive assertion method and its relation to computational induction on the following simple flowchart program:



We wish first to use the inductive assertions method to show that the above flowchart program over the natural numbers computes the factorial function, i.e., $z = x!$, *whenever it terminates*. To do this, we associate a predicate $Q(x, y_1, y_2)$, called an "inductive assertion," with the point labelled α in the program, and show that Q must be true for the values of the variables x, y_1, y_2 whenever execution of the program reaches point α. Thus, we must show (a) that the assertion holds when point α is first reached after starting execution, (i.e., that $Q(x, 0, 1)$ holds) and (b) that it remains true when one goes around the loop from α to α (i.e., that $y_1 \neq x \wedge Q(x, y_1, y_2)$ implies $Q(x, y_1+1, (y_1+1) \cdot y_2)$). To prove the desired result we finally show (c) that $z = x!$ follows from the assertion $Q(x, y_1, y_2)$ when the program terminates (i.e., that $y_1 = x \wedge Q(x, y_1, y_2)$ implies $y_2 = x!$).

We take $Q(x, y_1, y_2)$ *to be* $y_2 = y_1!$. Then:

(a) $Q(x, 0, 1)$ is $1 = 0!$.
(b) We assume $y_1 \neq x$ and $Q(x, y_1, y_2)$, i.e., $y_2 = y_1!$. Then $Q(x, y_1+1, (y_1+1) \cdot y_2)$ is $(y_1+1) \cdot y_2 = (y_1+1)!$, i.e., $(y_1+1) \cdot y_1! = (y_1+1)!$.
(c) We assume $y_1 = x$ and $Q(x, y_1, y_2)$, i.e., $y_2 = y_1!$, then $y_2 = y_1! = x!$ as desired.

To show the relation between this method and computational induction, we must first translate the flowchart program into a recursive program. Following the technique of McCarthy,[10] we find that the above flowchart program is $f_{P_5}(x, 0, 1)$, i.e., the least fixpoint of the recursive program P_5 (with $y_1=0$ and $y_2=1$) where

$$P_5: \quad F(x, y_1, y_2) \Leftarrow \textit{if } y_1=x \textit{ then } y_2 \quad \textit{else } F(x, y_1+1, (y_1+1)\cdot y_2).$$

We shall prove by (simple) computational induction that $f_{P_5}(x, 0, 1) \subseteq x!$, i.e., the value of $f_{P_5}(x, 0, 1)$ is $x!$ whenever $f_{P_5}(x, 0, 1)$ is defined, which is precisely what the above proof by inductive assertions showed.

We take $\varphi(F)$ to be the following predicate:

$$(\forall x, y_1, y_2)\{Q(x, y_1, y_2) \Rightarrow [F(x, y_1, y_2) \subseteq x!]\},$$

where $Q(x, y_1, y_2)$ is $y_2=y_1!$, the induction assertion used before. Obviously $\varphi(f^0)$ holds. To show that $\varphi(f^i)$ implies $\varphi(f^{i+1})$ for $i \geq 0$, we consider two cases: either $y_1=x$, in which case the proof follows directly from (c) above, or $y_1 \neq x$, which follows directly from (b).

By computational induction we therefore have $\varphi(f_P)$, i.e., $Q(x, y_1, y_2) \Rightarrow [f_P(x, y_1, y_2) \subseteq x!]$ for all x, y_1, y_2. But since $Q(x, 0, 1)$ is known from (a) to hold, we conclude that $f_P(x, 0, 1) \subseteq x!$, as desired.

REFERENCES

1 Z MANNA S NESS J VUILLEMIN
Inductive methods for proving properties of programs
In Proceedings of ACM Conference on Proving Assertions about Programs ACM New York January 1972

2 S C KLEENE
Introduction to meta-mathematics
Van Nostrand Princeton New Jersey 1952

3 J M CADIOU
Recursive definitions of partial functions and their computations
PhD Thesis Computer Science Dept Stanford University To appear

4 J VUILLEMIN
Proof techniques for recursive programs
PhD Thesis Computer Science Dept Stanford University To appear

5 R MILNER
Implementation and applications of Scott's logic for computable functions
In Proceedings of ACM Conference on Proving Assertions about Programs ACM New York January 1972

6 J W DEBAKKER D SCOTT
A theory of programs
Unpublished memo August 1969

7 J H MORRIS
Another recursion induction principle
CACM Vol 14 No 5 pp 351-354 May 1971

8 R W FLOYD
Assigning meanings to programs
In Proceedings of a Symposium in Applied Mathematics Vol 19 Mathematical Aspects of Computer Science pp 19-32 Ed J T Schwartz

9 P NAUR
Proof of algorithms by general snapshots
BIT Vol 6 pp 310-316 1966

10 J McCARTHY
Towards a mathematical science of computation
In Information Processing: Proceedings of IFIP 62 pp 21-28 Ed C M Popplewell Amsterdam North Holland 1963

Mathematical concepts in programming language semantics

by DANA SCOTT

Princeton University
Princeton, New Jersey

INTRODUCTION

In mathematics after some centuries of development the semantical situation is very clean. This may not be surprising, as the subject attracts people who enjoy clarity, generality, and neatness. On the one hand we have our concepts of mathematical *objects* (numbers, relations, functions, sets), and on the other we have various formal means of *expression.* The mathematical expressions are generated for the most part in a very regular manner, and every effort is made to supply all expressions with denotations. (This is not always so easy to do. The theory of distributions, for example, provided a non-obvious construction of denotations for expressions of an operational calculus. The derivative operator was well serviced, but one still cannot multiply two distributions.)

The point of such activity is that the formal rules of the calculus of expressions allow solutions of problems to be found and give us ways of checking correctness of proposed solutions. It is by no means the case that mathematical inspiration is thus reduced to automatic algebra, but the possibility of formal manipulation is a great help. Not only can we record useful lemmas, but a precise language is essential for teaching and communication.

It is of course possible to pick formalisms out of thin air and to ask people to accept their usefulness on mystical grounds. (The operational calculus was developed a little that way.) There is no denying that clever guesswork can be very successful, but ultimately it is fair to ask the mystic just what he is talking about. A way to counter that question is to show how the concepts being symbolized in the new language are explained in terms of familiar notions. To do this we can try either to correlate directly denotations definable in familiar terms with the expressions of the new language or to show how to translate the new expressions out of every context in which they might occur leaving only what is familiar remaining. If you can do it the first way, you can do it the second way, obviously; the converse is not obvious though. A contextual translation scheme may be very sensitive to the context: a transformation appropriate for one occurrence of an expression may not work elsewhere.

The *negative integers* can be used as a good example here. The rules of algebra are such that all minus signs can be eliminated from an equation by multiplying out the formal polynomials and then using transposition. (Different rules are used for different contexts, note.) If it is the question of an *algebraic law* with variables, we use this trick: Suppose we wish to assert that the equation $f(x)=g(x)$ holds for all x at both positive and negative values. Of course $f(x)=g(x)$ holds for all non-negative x. But so does $f(-x)=g(-x)$. In other words *one* equation has been equivalently translated into *two,* in which the variable can be restricted to the more familiar range. Now Descartes may have been willing to do algebra this way, but we would no longer consider it reasonable. We have developed our theory of numbers so as to include both positive and negative quantities, and we have gone on to imaginary and complex numbers with great gains in understanding. In the new algebra, expressions are directly meaningful and no translations are required. Nevertheless we prove that the new algebra is *consistent* by constructing a model out of familiar stuff. Complex numbers were fairly easy to handle; quaternions were harder. You all know examples and know that mathematicians are busy every day finding other weird structures. Whether the results deserve other than a mother's love is another question.

What does this discussion have to do with programming languages? Very much I feel. Programming languages have introduced a new dimension into the problem of semantics. In the first place there has been an explosion in the size and complexity of the expressions that must be considered. In view of the complications, every effort is made to allow the writer of the program to keep tiresome details *implicit*: some controls can be written in, but generally it is the *compiler* that will take care of the boring parts. Now many, many compilers have been written—even students can do it these days. A compiler can be viewed as a *translator* from the source language (unfamiliar) into machine language (familiar). Why cannot we say, then, that the semantical problems of programming language are solved? The answer is, I believe, that every compiler (if it is meant for a real machine) is a compromise. There must be as many compilers as there are machines, each reflecting limitations and peculiarities of the given machine. Should not the ideal of language definition be machine independent?

Certainly the introduction of *abstract* machines does not settle this difficulty of a multiplicity of translation schemes. After all even for an abstract machine the compiler writer must choose specific ways of keeping track of the order or depth of nesting of computation initiations. There may be different ways of doing this, all of which lead to the same results. The variety of approaches even on an abstract machine may make it quite impossible to define in any meaningful way just what is a single computation *step*. Therefore, the idea of the *state of the computation* is not apparently at all well determined by the source language alone. When a specific compiler is fixed, however, we may very well want to investigate how it handles computation steps. But the question remains: do they have anything to do with the semantical definition of the original language? I think not.

Before pursuing the question of the proper level at which semantics enters, we should take note of a second dimension-expanding consequence of programming language study. Mathematical concepts generally have a rather *static* quality. True, mathematicians study flows on manifolds and many-body problems, but it seems to me that often the interest centers on global qualities or asymptotic behavior. What is peculiar in programming language—especially from a linguistic point of view—is that exactly the same symbols may possess a quite different import in different segments of a program. This is most apparent in the way variables (identifiers) are used. I need give no examples here; all of you understand why variables are not employed in the *assignment statement* in a "mathematical" way. The dynamic character of command execution requires a quite different treatment of variables, and this dynamic urge permeates the whole semantic analysis. The principal virtue of the programming language is that the compounding of the various operations need only be indicated in a rather schematic way, since the calculation is meant to be automatic in any case. But the calculation is not meant as a game or a joke. It is something that needs precision, so we must be able to "follow" what happens in some rather definite sense. What we are discussing is how definite this has to be in order to have a good semantical definition of a language.

Some years ago, Christopher Strachey began a project of trying to expose the semantics of a programming language as a series of mutually recursive functional equations.[1] The particular complex of equations would be "syntax directed" in the sense that each clause of the recursive syntactical specification of the language would have a corresponding equation. The equations taken together would define the meaning of a program, because made to correspond to a program text would be a recursively defined function. This function was intended as a state-transformation function: the function that takes an initial state to the final state that results from executing the program. In many instances the notion of "state" used here could be taken to be "state of the memory" or some abstraction from that idea to make the definition machine independent.

The formalism Strachey used to state his equations was a modified version of the λ-calculus. There was an inherent difficulty in this approach: The λ-calculus at the time was just another symbolic calculus.[2] The criteria for equivalence of expressions were rather weak, and it was not clear in which directions they should be expanded. The advantage of the λ-calculus was that it was "clean," especially in its handling of variables, which was done along the usual "mathematical" lines.[3,4] Thus it was felt that some progress had been made in explaining the unfamiliar in terms of the familiar. Still, the project was not quite complete.

One objection to the λ-calculus the present author had was that it possessed no *mathematical* semantics. It was another kind of operational calculus. True, the calculus had been proved *consistent*, but the consistency proof did not seem to give clear hints as to how to extend usefully the principles of equivalence of expressions.[5] This uncertainty was particularly unsettling in view of the many *inconsistent* extensions that had been proposed. The main difficulty rested on the thoroughly *type-free* style of functional application and abstraction used in the λ-calculus, where, for example, a function could be applied to *itself* as an argument.[6] And it was just this type-free character of the system that Strachey wanted to exploit.

During the fall of 1969 the author resided in Oxford in order to learn more about Strachey's ideas on programming languages. After long, and sometimes rather heated, discussions on the role of mathematics in semantics, it slowly began to dawn on the author that there might be an "objective" interpretation of the λ-calculus. As explained elsewhere,[7,8] it was possible finally to combine ideas and techniques from many sources into a coherent theory of models for the λ-calculus. The key step was to find the right idea of an abstract *domain* (which turned out to be a kind of topological space) so that an appropriate notion of a *function space* would be a domain of the same kind. From that point it was only a matter of time until the question of whether a domain might exist that could be isomorphically identified with its own function space would arise. Fortunately the author was able to see how to effect the required mathematical construction, and a new approach to functional calculi was opened up.

This sounds most abstract and impractical, but the spaces constructed are actually no worse then the real numbers. The "irrational" elements can be explained as limits of infinite sequences in ways that are quite reminiscent of classical analysis.[9] And what is more important *proofs* about the properties of the elements can be given inductively by "successive approximation" in a simple and natural way. Furthermore it is possible to discuss the effectiveness and computability of various elements and processes so as to tie in the abstractions with actual computation.

In this paper it will not be possible to discuss the mathematical details, though a few words will be inserted here and there to resolve puzzles about whether the method is reasonable.[10] Instead it will be shown how the theory of these abstract domains can be applied to typical semantical problems, thus supplying the mathematical foundation needed for Strachey's equational approach. It is not claimed that the project for a mathematical semantics is at all completely finished, but the ideas have been found to be so flexible that we have gained full confidence that we can do serious model building incorporating features whenever the need for them is realized.[11]

LOCATIONS AND STORES

There are many levels of storage from core to disk to tape to card hopper to the mind of the user sitting at a console. Some of these are more addressible than others (and more reliable). Some require special handling, especially as regards allocation of space in preparation for future acts of storage. In the present discussion we shall not treat these subtleties, even though they are very important for good compiler operation. We shall simplify the presentation for the sake of illustration by making these assumptions:

- There are but two facets to storage: *internal* and *external*.
- The internal storage is addressed by elements of a homogeneous collection of *locations*.
- The *contents* stored at each location always may be read out (nondestructively), and at any time new contents may be read in *updating* (destroying) the old contents.
- The current *state* of the store allows a distinction between the *area* of active locations (with contents) and passive or free locations (without contents).
- At any time a *new* location may be selected from those free and placed in the active area.
- At any time a location may be *lost* or retired to the free list.
- One part of the external store allows *reading* in of fresh information (input).
- The other part of the external store allows *writing* of computed information (output).
- External reading and writing are *independent* operations: what is written cannot be read back unless it has been separately stored internally.

The statements of these rather natural assumptions have been given first in fairly precise yet intuitive language. The point of the present paper is to show how to "mathematize" such assumptions in terms of a model that can be incorporated into a full semantical definition. (No *formal language* has been discussed at this point, however.) To carry out the mathematical modelling, we postulate first the existence of several *domains* that will be interrelated by the model:

The basic domains

L = locations

V = stored values

S = states of the store

T = truth values

Possibly it would have been better to say "storable values" since not every element of V is actually stored. Just what these quantities are supposed to be will be left open until the next section. One has a variety of choices depending on the kind of language that is to be interpreted.

These domains alone do not specify the model because they have not yet been supplied with any structure. This can be effected by listing the operations and transformations that may be applied to the domains in various combinations:

The store operators

$$\begin{aligned} \textit{Contents}&: \mathrm{L}\to[\mathrm{S}\to\mathrm{V}] \\ \textit{Update}&: \mathrm{L}\times\mathrm{V}\to[\mathrm{S}\to\mathrm{S}] \\ \textit{Area}&: \mathrm{L}\to[\mathrm{S}\to\mathrm{T}] \\ \textit{New}&: \mathrm{S}\to\mathrm{L} \\ \textit{Lose}&: \mathrm{L}\to[\mathrm{S}\to\mathrm{S}] \\ \textit{Read}&: \mathrm{S}\to\mathrm{V}\times\mathrm{S} \\ \textit{Write}&: \mathrm{V}\to[\mathrm{S}\to\mathrm{S}] \end{aligned}$$

Given any two domains D_0 and D_1, we write $[D_0\to D_1]$ for the space of all (admissible) functions from D_0 into D_1. (At this stage we do not need to worry just which functions are admissible.) We write $f: D_0\to D_1$ to indicate that f is just such a function. What we have listed above are the functions that correspond to our intuitive ideas about the structure of the store. In order to explain how this works, the following variables will be used (with or without sub- and superscripts) restricted to the indicated domains:

$$\alpha{:}\mathrm{L},\quad \beta{:}\mathrm{V},\quad \sigma{:}\mathrm{S},\quad \tau{:}\mathrm{T}$$

Suppose, then, α is a particular location and σ is the current state of the store. The function value *Contents*$(\alpha)(\sigma)$ is to be the V-value stored by σ at α. Conversely, let β be a *given* V-value. The *pair* (α, β) in L×V provides the desired information for updating the store. Therefore, when we write

$$\textit{Update}(\alpha, \beta)(\sigma) = \sigma'$$

we want σ', the transformed store, to be just like σ except in having β stored now in location α. Similarly *Area*$(\alpha)(\sigma)$ is *true* or *false* according as α is in the active area of σ. The function *Lose*(σ) transforms σ into the store σ'' which is just like σ except that α has been freed. Exactly what *Read* and *Write* do depends on the activities of the mad scientist at the console. Supposing, however, that he does not ruin the equipment, whenever *Read*(σ) is activated *something* will happen. This transformation can be recorded as a pair (β', σ'), where β' is a storable value, and σ' has the same internal state as σ did, but the external condition is made all eager and ready for the next input. In a similar way, the function *Write*(β') may be regarded as only affecting the external portion of the store.

The functional notation of mathematics has the gloss of precision. But so far the gloss is the only aspect introduced, since note that the above explanations of what the functions are supposed to do are just as informal as the original assumptions given in ordinary language. What must be avoided in this kind of work is the Fairthorne *Hovercraft effect* described[12] as an "approach in which we use extremely sophisticated techniques and mathematical devices to skim at very high speed and great expense over the subject of concern without ever touching it."

Now in mathematical model building the way to touch ground is to find out how to formulate with precision the *postulates* that correspond to the intuitive assumptions. *Notational* precision is not enough—if there is no way of proving useful theorems. So far our postulates have been rather weak: nothing more than simply noting the functional characters (or logical *types*) of certain operators. The next step is to say exactly how these operators are connected.

What I wish to claim is that each of the intuitive assumptions previously informally indicated can be expressed rigorously by an *equation.* For example, after updating the store at a location, the contents of the location will be found to be what is expected and the location will be found to be active. This statement makes two equations as follows:[r]

$$\textit{Contents}(\alpha)(\textit{Update}(\alpha, \beta)(\sigma)) = \beta$$

$$\textit{Area}(\alpha)(\textit{Update}(\alpha, \beta)(\sigma)) = \textit{true}$$

These equations are meant to hold for *all* α, β, σ. What about locations α' different from α? If $\alpha'\neq\alpha$, we expect that:

$$\textit{Contents}(\alpha')(\textit{Update}(\alpha, \beta)(\sigma)) = \textit{Contents}(\alpha')(\sigma)$$

Similarly in the case of *Area.* If we introduce the *equality relation* on L:

$$\textit{Eq}: \mathrm{L}\times\mathrm{L}\to\mathrm{T}$$

and *conditional expressions,* these four equations can be reduced to two in the obvious way.

In writing these equations, one puzzle is what to write in the cases where the functions are possibly *undefined.* A solution is to introduce a *special value* to correspond to this case. The symbol for this value adopted here is $\perp$. We write $f(x) = \perp$ to indicate that the function x is undefined at x. (Other symbols in use are Ω, ω, $*$, and UU.) This convention is useful in the main equation relating *Contents* to *Area*:

$$\begin{aligned} \textit{Contents}(\alpha)(\sigma) = \textit{if } & \textit{Area}(\alpha)(\sigma) \\ & \textit{then Contents}(\alpha)(\sigma) \\ & \textit{else } \perp \end{aligned}$$

There is not enough room here to formulate all the necessary equations; thus, my claim must remain just that.[13] But it does not seem to me that the claim is too implausible. Note, however, that these postulates on the character of the store would not completely specify everything. The exact nature of V is left open at this stage of the model building. Concerning L, all we would know is that there are infinitely many locations, because new ones can be produced indefinitely. Actually that is probably enough to know about L until one gets into problems of allocation. The input/output phase has also been left a little vague, but that is not necessarily bad either. If we can prove any theorems at all, it is better to prove them from fewer assumptions. *Some* assumptions are needed, nevertheless; and it requires experimentation to find just the right balance. Part of the thesis of this paper is that the mathematical style advocated allows for this experimentation to take place in a convenient way.

A LANGUAGE

There were no surprises in the discussion of storage in the previous section, and there are no surprises planned for this section either. All we shall do here is to indulge in some very modest language design. Any real innovation is saved for the section on semantics. And this is how it should be, otherwise we would not be giving solutions to natural problems. It is quite possible, though, that after the semantical approach has been appreciated language design practice will be altered. Strachey has argued for this,[14] but the effects remain to be seen. In any case we hope to develop a semantical style within which the consequences of different design choices can be studied.

To start with our language will be divided into the usual syntactical categories:

The syntactical domains

$$\text{Id} = \text{identifiers}$$

$$\text{Num} = \text{numerals}$$

$$\text{Op} = \text{(binary numerical) operations}$$

$$\text{Cmd} = \text{commands}$$

$$\text{Exp} = \text{expressions}$$

A feature of our method is that we treat these domains on a par with the basic and semantical domains. They are different, since they are constructed for syntactical purposes; but they are domains nevertheless. The following variables are used in restricted senses:

$$\xi{:}\text{Id}, \quad \nu{:}\text{Num}, \quad \omega{:}\text{Op}, \quad \gamma{:}\text{Cmd}, \quad \epsilon{:}\text{Exp}$$

The syntax for Id, Num, and Op will not be given since it is standard. The remaining two categories are defined as follows:

Commands

$$\gamma ::= \epsilon_0 := \epsilon_1 \mid \text{write } \epsilon \mid !\epsilon \mid \epsilon \rightarrow \gamma_0, \gamma_1 \mid$$

$$\text{dummy} \mid \gamma_0 ; \gamma_1 \mid \text{while } \epsilon \text{ do } \gamma \mid$$

$$\text{let } \xi \equiv \epsilon \text{ in } \gamma \mid \text{let } \xi = \epsilon \text{ in } \gamma \mid \text{letrec } \xi \text{ be } \epsilon \text{ in } \gamma \mid$$

$$(\gamma)$$

Expressions

$$\epsilon ::= \xi \mid \perp \mid \text{read} \mid \gamma \text{ result is } \epsilon \mid \epsilon_0 \rightarrow \epsilon_1, \epsilon_2 \mid$$

$$\epsilon{:}\text{T} \mid \text{true} \mid \text{false} \mid \epsilon_0 = \epsilon_1 \mid$$

$$\epsilon{:}\text{N} \mid \nu \mid \epsilon_0 \omega \epsilon_1 \mid$$

$$\epsilon{:}\text{C} \mid {:}\gamma \mid$$

$$\epsilon{:}\text{P} \mid \lambda\xi . \epsilon \mid \epsilon_0 \epsilon_1 \mid$$

$$\epsilon{:}\text{R} \mid \downarrow \epsilon \mid \uparrow \epsilon \mid$$

$$(\epsilon)$$

Some comments are necessary: (i) There is an unpleasant *conflict* in this syntax between the use of words versus the use of symbols. The trouble is that there are not enough symbols—especially on keyboards—otherwise all constructs could be symbolized. But then one has to remember what the symbols are for, so it is better to use words—especially English words. But it takes so long to write out all those words. The situation is hopeless. (ii) There is an unpleasant *ambiguity* in this syntax. The same string could be parsed in several different ways. Necessarily, then, the meaning of a command or an expression will have to depend on the *parse.* But we will pretend in writing equations that everything can be determined from the string alone. If that worries you, please write in the parentheses (you are allowed to by the last clause of the definition). But no one can stand to write in all those parentheses. The situation is hopeless. But for the semantical theory it does not really matter.[15]

The style of language definition was chosen to give a *few* hints as to the meanings. But I hope there are some constructs that seem quite mysterious on a first reading.

Some explanation is in order. Take commands first. The first clause gives us *assignment statements.* Everyone knows what they mean—except what does it mean to have a *compound* expression on the left? Well, never mind. The next command is obviously intended to invoke *output.* That was easy. For the next clause it would have been better to write do ϵ. In this language commands can be set up and (their closures) stored away for later execution. This command means that ϵ should be evaluated out; and if the value is a command closure, then it should be *done.* If not, what then? Never mind. Next we have the *conditional* command (Boolean-valued expressions are possible). To execute dummy, *do nothing*—except waste time. The semicolon provides the means for sequencing, or *composition,* of commands. Next we have the while statement. Its effect is well-known. Finally there are three kinds of initialization of parameters in blocks. But which is which?

An odd fact about many programming languages is that identifiers are used for many different purposes even though they form just one syntactical category. The context is expected to give the clue as to use. It is not such a bad idea. When one starts to write a program, the number of identifiers to be used is probably not known. To have to divide them up into various subcategories would complicate the language a great deal, and restrict the ways they could be chosen, by tiresome rules. They have only a local significance in any case, so some rough, informal conventions of programming style are sufficient to assure readability of programs.

The three types of initialization in this language illustrate three quite different uses of identifiers. In the first, we begin by evaluating ϵ. It will turn out to be either an L-value or a V-value. Whatever it is, we associate the *same* value with ξ and go on to execute γ under this assignment (keeping assignments to other identifiers the same as before, of course). In the second, we evaluate ϵ. If the value is a V-value, we are happy. If the value is an L-value, we find the contents of the current state of the store in that location—a V-value. In either case we now have a V-value. Next we find a *new* L-value outside the active area of the store and attach it to ξ. Finally we update this location with the computed V-value. Thus prepared, we can now execute γ. (The only reason for being so complicated is that in γ we would like to be able to use ξ on the left of assignment statements.) The first two cases are not so very different, but the third presents a fresh aspect.

At the time of making a *recursive* definition, we want to associate the "import" of ϵ with ξ, but we do *not* want to evaluate ϵ. Also ϵ may contain ξ (a free occurrence of the identifier), and we want ξ to mean the same as we just said. Having made this tie-up without actually evaluating anything, we begin to execute γ. Every time we find a (free) ξ, in effect, we want to *replace* it be ϵ and continue the evaluation. That is to say, ϵ will be evaluated only when it is called for by an occurrence of ξ. Since ϵ contains ξ, this may happen repeatedly. This sequence of events is quite unlike the non-recursive initializations where the ξ's in ϵ are not at the same as the ξ's in γ.

Turning now to expressions, the explanations can be rather short. The read expression invokes input and has a V-value. The result is combination allows for the command γ to be executed before ϵ is evaluated. Next we see the *conditional* expression. The ϵ:X expressions are all Boolean-valued and are *tests* on the nature of the value of ϵ. In lines two and three of the definition we have obvious *Boolean-valued* or *numerical-valued* expressions. In line four we have to do with (closures of) *commands,* and :γ forms just such a closure making a command into an expression. Line five has to do with *procedures.* First we find λ-*abstraction* (functional abstraction) which forms a (closure of) a procedure from an expression with respect to a formal parameter. Next we have *application* of a procedure to an argument. Line six, finally, has to do with *references.* We can *make* a reference (to the value of ϵ), or we can *look up* a reference. References bring in the problem of *sharing,* but we will not treat it in this paper. Nor shall we discuss *arrays.* Note too that *labels* and *jumps* have not been included.

The explanations just given have not been very full; but the programming concepts mentioned are familiar, and most of us would now have a fairly good idea of what a program written in this language would do. Such intuitive understanding is not rigorous semantics, however, and as everyone knows sticky points can arise. Just as we could turn the intuitive assumptions about the store into precise equations, so must we formalize our semantical understanding, as will be outlined in the next section.

THE SEMANTICS

The plan of the semantical definition is that every command and every expression shall denote something. It is the construction of the domains in which the values lie that requires the mathematics. We have spoken already about the basic domains L, V, S, and T. Of these L and T can be given independently in advance, but V and S can only be formed in conjunction with the over-all semantical project. To do this we have to look at the kinds of expressions there are to be found in the language and to provide appropriate domains. The uses of the identifiers also have to be taken into account. This analysis suggests these domains in addition to the four already isolated:

The semantical domains

N = numerical values

E = expression values

W = dynamic values

D = denoted values

C = command values

P = procedure values

R = reference values

The denoted values accumulate all the possible values that might be attached to an identifier. A scheme of such attachments is called an *environment*. Mathematically it is just a function from identifiers to denoted values. All our evaluations will have to take place within (or relative to) such an environment. The following variable is restricted to environments:

$$\rho\colon \mathrm{Id} \to \mathrm{D}$$

Environments can change, but the meanings of the numerals and the numerical operations are fixed. The way to fix them is to define in the standard way these two functions:

$$\mathit{Numval}\colon \mathrm{Num} \to \mathrm{N}$$

$$\mathit{Opval}\colon \mathrm{Op} \to [\mathrm{N} \times \mathrm{N} \to \mathrm{N}]$$

Explicit definitions need not be given here.

Ultimately a command means a store transformation. In mathematics expressions have "static" expression values, but in programming it turns out that commands can be invoked in evaluating a command. Thus an expression has an *effect* as well as a value. We also have to remember that expressions are sometimes used for their L-values and sometimes for their V-values. All of this can be summarized in several semantical functions which have these logical types:

The semantical functions

$$\mathcal{C}\colon \mathrm{Cmd} \to [[\mathrm{Id} \to \mathrm{D}] \to [\mathrm{S} \to \mathrm{S}]]$$

$$\mathcal{E}\colon \mathrm{Exp} \to [[\mathrm{Id} \to \mathrm{D}] \to [\mathrm{S} \to \mathrm{E} \times \mathrm{S}]]$$

$$\mathcal{L}\colon \mathrm{Exp} \to [[\mathrm{Id} \to \mathrm{D}] \to [\mathrm{S} \to \mathrm{L} \times \mathrm{S}]]$$

$$\mathcal{V}\colon \mathrm{Exp} \to [[\mathrm{Id} \to \mathrm{D}] \to [\mathrm{S} \to \mathrm{V} \times \mathrm{S}]]$$

To understand how these functional types are arrived at, consider a command γ. Given an environment ρ and a state of the store σ, we can write the equation

$$\mathcal{C}(\gamma)(\rho)(\sigma) = \sigma'$$

to indicate that σ' will be the state achieved after execution of γ. Similarly for expressions the equation

$$\mathcal{E}(\epsilon)(\rho)(\sigma) = (\eta, \sigma')$$

indicates that η is the value found for ϵ, while σ' is the state achieved after the act of evaluation.

An immediate question is why the ρ and σ are separated as arguments. The answer is that they enter into the evaluations in different ways. For one thing, they change at quite different rates with ρ staying fixed much longer than σ. For another, we will be tempted to copy ρ, but we will generally never feel free to ask for a copy of the *whole* store—there just is no room for that. Possibly in simulating a small machine on a large machine we could conceive of programming the copying of the store, but this is not yet a standard concept. In a way ρ enters at a more conscious level, while σ flows along as the unconscious. Much of the action on σ is meant to be automatic, though there is communication between the levels. Besides these reasons, it is often the case that we want to discuss $\mathcal{C}(\gamma)(\rho)$ without reference to a particular σ; thus it is convenient to be able to drop it off as the last (deepest) argument.

For stylistic reasons and as an aid to the eye, we shall use *emphatic brackets* with the semantical functions:

$$\mathcal{E}[\![\epsilon]\!](\rho)(\sigma)$$

These are helpful because the expressions tend to get along and to pile up their own parentheses. Before we can discuss the semantical equations, however, we must return to the definitions of the various domains.

The domains L, T and N are given. The rest are constructed. The constructions can be summarized as follows:

The domain equations

$$\mathrm{V} = \mathrm{T} + \mathrm{N} + \mathrm{C} + \mathrm{P} + \mathrm{R}$$

$$\mathrm{E} = \mathrm{V} + \mathrm{L}$$

$$\mathrm{W} = \mathrm{S} \to \mathrm{E} \times \mathrm{S}$$

$$\mathrm{D} = \mathrm{E} + \mathrm{W}$$

$$\mathrm{C} = \mathrm{S} \to \mathrm{S}$$

$$\mathrm{P} = \mathrm{E} \to \mathrm{W}$$

$$\mathrm{R} = \mathrm{L}$$

We shall not need variables over all of these domains. The following restricted variables are used:

$$\eta{:}\mathrm{E},\quad \delta{:}\mathrm{W},\quad \theta{:}\mathrm{C}$$

One domain is missing from the list: namely, S. The construction of S is a bit complicated and needs more discussion of L than we have time for here. Roughly

$$\mathrm{S}=\mathrm{A}\times[\mathrm{L}\to\mathrm{V}]\times\mathrm{I/O},$$

where the A-coordinate represents the area; the middle coordinate, the contents-function; and the last, the input/output features. But this is a little too rough, though we must leave it at that.[16]

What is essential to note here is the highly recursive character of these domain equations; in particular, V involves S and S involves V. The involvement comes about through the use of function domains such as [S→S] and [L→V]. Since V-values are storable values, something seems wrong here. We said earlier that we would not store (i.e., copy) stores, yet we are asking to store a store *function* $\theta{:}\mathrm{S}\to\mathrm{S}$. There is actually no conflict here. Functions involve their arguments in a *potential*, not actual, manner. In implementing such a language, functions would be stored as finite *descriptions* (i.e., texts of definitions) not as infinitary abstract objects. From the point of view of mathematical conception, however, we want to imagine in theory that it is the functions that are stored. If we did not want to do this, we should explicitly program the syntax of function definition and the necessary interpretive operations. In any case, storing a function is done without storing the store.

Granting the reasonableness of the connections between V and S, it is still fair to ask whether such domains exist. This is where the real mathematics enters. The idea mentioned in the Introduction of constructing λ-calculus models can be employed to construct these more involved domains. It requires a combination of lattice theory and topology to achieve the proper kind of function theory.[17] On set-theoretical grounds it is clear that when we write [S→S] we cannot mean to take *arbitrary* functions, otherwise the cardinality of S (and of V) would become unbounded. The point is that the functions defined by programs are of a very special sort. Looked at in the right way they are *continuous* functions, and there are many fewer continuous functions than there are arbitrary functions. And this turned out to be the secret: by restricting the function spaces to the continuous functions, the required domains can indeed be constructed. Expensive as this device is, the Hovercraft effect is avoided by showing that such a theory of domains fits in very well with the theory of computable functions (recursive functions) and that all the functions required by the semantics of the language are indeed computable. In particular the theory provides a basis for justifying the existence of functions defined by involved recursions (like the semantical functions) and for proving properties of such functions.[18,19]

It should be stressed that the semantical domains make no explicit reference to the syntax of the language. True, we gave the language first, and then cooked up the domains to match. It could have easily been the other way round. Take V for example. There were five main sections to the definition of an expression corresponding to five sorts of values an expression could have. (Strictly speaking this is not true. The expressions $\epsilon_0\epsilon_1$ and $\uparrow\epsilon$ could have arbitrary values. It might have been more logical to put these clauses on the first line of the definition.) Therefore in the construction of V, we had to provide for five sorts of elements, T, N, C, P, and R. The summation operation (+) on domains has to be defined as a *disjoint union* (and a little care is needed with the topology of the resulting space). If we had wanted, we could have included other sorts of summands. They would not have been reflected in the language, however. It is an interesting question whether the language reflects enough of the structure of the domain V, but one that we cannot pause to consider here. The point is that language design and domain construction are parallel activities, each can (and should) affect the other.

Finally it remains to illustrate some of the equations needed to define the semantical functions:

Some semantical equations

$$\mathcal{C}[\epsilon_0{:}=\epsilon_1](\rho)=\mathit{Update}{*}\mathit{Pair}(\mathcal{L}[\epsilon_0](\rho))(\mathcal{V}[\epsilon_1](\rho))$$

$$\mathcal{C}[\mathrm{write}\ \epsilon](\rho)=\mathit{Write}{*}\mathcal{E}[\epsilon](\rho)$$

$$\mathcal{C}[\gamma_0;\gamma_1](\rho)=\mathcal{C}[\gamma_1](\rho)\circ\mathcal{C}[\gamma_0](\rho)$$

$$\mathcal{C}[!\epsilon](\rho)=\mathit{Do}{*}\mathcal{E}[\epsilon](\rho)$$

$$\mathcal{C}[\mathrm{let}\ \xi\equiv\epsilon\ \mathrm{in}\ \gamma](\rho)=(\lambda\eta\cdot\mathcal{C}[\gamma](\rho[\eta/\xi]))*\xi[\epsilon](\rho)$$

$$\mathcal{C}[\mathrm{letrec}\ \xi\ \mathrm{be}\ \epsilon\ \mathrm{in}\ \gamma](\rho)$$

$$=\mathcal{C}[\gamma](\rho[\mathit{Fixpoint}(\lambda\delta\cdot\mathcal{E}[\epsilon](\rho[\delta/\xi]))/\xi])$$

For the $\mathcal{C}$-function there should be 11 equations in all; and for the $\mathcal{E}$-function, 21 equations. Then the semantical definition would be complete. Note how the equations are mutually recursive functional equations: the $\mathcal{C}$- and $\mathcal{E}$-functions occur on both sides of the equations in functional contexts. Note too that there is much unexplained notation.

In order to be able to combine functions, we need at least two kinds of composition; because some functions are of type [S→S], and some are of type [S→X×S]. We can see how this works in one example. Let:

$$f = \mathcal{L}[\epsilon_0](\rho) : \mathrm{S} \to \mathrm{L} \times \mathrm{S}$$

$$g = \mathcal{V}[\epsilon_1](\rho) : \mathrm{S} \to \mathrm{V} \times \mathrm{S}$$

We want:

$$\mathit{Pair}(f)(g) : \mathrm{S} \to [\mathrm{L} \times \mathrm{V}] \times \mathrm{S}.$$

Suppose σ is a given state. Then $f(\sigma) = (\alpha, \sigma')$. Carrying over σ' to the next step, $g(\sigma') = (\beta, \sigma'')$. So we can define:

$$\mathit{Pair}(f)(g)(\sigma) = ((\alpha, \beta), \sigma'').$$

(The reason for doing it this way is to get the effect of two separate evaluations, because there were two expressions ϵ_0 and ϵ_1 needing evaluation.) Then by definition:

$$(\mathit{Update} * \mathit{Pair}(f)(g))(\sigma) = \mathit{Update}(\alpha, \beta)(\sigma'')$$

This shows how $*$ is a form of functional composition. In the third equation we wrote $\circ$, since this was just the ordinary composition of two [S→S] functions.

The notation $\rho[\eta/\xi]$ means that the original environment ρ has been altered to assign the value η to the argument ξ. (Remember that the arguments of environments are the formal identifiers as syntactic objects.) The import of the fifth equation then is to evaluate ϵ first, pass on the value to the environment, and then execute the command starting in the state achieved after evaluation of ϵ. Note that ϵ is evaluated in the original environment.

The *Fixpoint* function in the last equation is applied to a function F: W→W. All our domains are constructed so that functions defined by expressions always have fixed points—*if* they have the same domain of definition and range of values, of course. The domains are lattices as well as topological spaces, so we can speak of the *least* solution to the equation:

$$\delta_0 = F(\delta_0).$$

This $\delta_0 = \mathit{Fixpoint}(F)$. Fixed points of functional equations solve recursions. So in the letrec command, we first set up the solution δ_0, and then execute the command in the environment $\rho[\delta_0/\xi]$. This explains our *mathematical* concept of recursion.[20] Note that there are no stacks, no pointers, no bookkeeping. None of those devices of implementation are relevant to the concept. That is why this is a mathematical semantics.

CONCLUSION

The presentation in this paper has been but a sketch. For a complete semantics even of this small language, many more details would be required. But it is hoped that enough of the project has been exposed to convince the reader that it is possible to carry out such a plan. And it is not even too painful to do so: not so many mathematical constructs are required, and the equations need not be so very complicated. Of course, judgment of the worth of doing so should be withheld until the semantical equations for a really serious language are fully written out.

REFERENCES

1 C STRACHEY
Towards a formal semantics
In T B Steel (ed) Formal Language Description Languages (North-Holland 1966) pp 198-220. Cf. also Ref 15 for another description of the present program

2 H B CURRY ET AL
Combinatory logic
Vol 1 (North-Holland 1958) Vol 2 (North-Holland 1972 in press) This is a basic reference with a very large bibliography

3 P J LANDON
The mechanical evaluation of expressions
Comp J 6 pp 308-320 1964

4 A EVANS JR
PAL—a language for teaching programming linguists
In Proc ACM 23rd Nat Conf Brandon Systems Princeton NJ The PAL language carries out many of the ideas introduced by Landon

5 Cf. Ref 2 p 78

6 The first proof of inconsistency was given by Kleene and Rosser for a system of Church. The proof was later much simplified. Cf. Ref 2 pp 258 ff

7 D SCOTT
Outline of a mathematical theory of computation
Proc Fourth Annual Princeton Conf on Info Sci and Sys Princeton 1970 pp 169-176

8 D SCOTT
Continuous lattices
Proc Dalhousie Conf Springer Lecture Notes in Math 1972 in press

9 D SCOTT
The lattice of flow diagrams
In E Engeler (ed) Semantics of Algorithmic Languages Springer Lecture Notes in Math Vol 188 1971 pp 311-366. This paper provides many examples of the limit concept. Cf. also Ref 10

10 D SCOTT
Lattice theory, data types and semantics
NYU Symp on Formal Semantics Prentice-Hall 1972 in press. This is a general introduction more detailed in some respects than Ref 7

11 C STRACHEY
Varieties of programming language
Proc ACM Conf Venice 1972 to appear. This paper contains more examples

12 R A FAIRTHORNE
Remarks in P Atherton (ed) Classification Research Munksgaard Copenhagen 1965 p 210

13 C STRACHEY
An abstract model of storage
In preparation. This report will give complete details

14 Cf. Ref 11

15 D SCOTT C STRACHEY
Toward a mathematical semantics for computer languages
Proc of Symp on Computers and Automata Microwave Research Institute Symp Ser Vol 21 Polytechnic Institute of Brooklyn 1972 in press. A nearly identical language and its semantics is outlined in this paper together with some more of the mathematical background

16 Cf. Ref 13

17 Cf. Refs 7 and 8

18 R MILNER
Implementation and applications of Scott's logic for computable functions
SIGPLAN Notices Vol 7 No 1 January 1972 pp 1-6. (=SIGACT News No 14)

19 J W De BAKKER
Recursive procedures
Math Centre Tracts 24 Mathematisch Centrum Amsterdam 1971. Other useful references are contained in these two papers

20 Cf. Refs 9, 15 and 19 for further discussion of recursion

Applications of language theory to compiler design

by J. D. ULLMAN

Princeton University
Princeton, New Jersey

INTRODUCTION

There are certain aspects of language theory that have had, or can have, significant impact on the design and implementation of compilers. These areas are, principally, the subjects of context free grammars and syntax directed translations. It is perhaps to be expected that the deep and interesting theorems of language theory do not usually find application. Rather, it is the definitions of formal constructs and their elementary properties that find use.

In this paper, we will discuss some of the ways in which the language theorist can help the compiler designer. First, we will discuss classes of context free grammars that have fast parsing algorithms and discuss briefly what these algorithms are. Then we shall discuss schemes for specifying the translations which must be performed by the compiler.

CONTEXT FREE GRAMMARS

Let us proceed to the principal formalism of use to compiler writers, the context free grammar.

Definition: A *context free grammar* (CFG) is a four-tuple (N, Σ, P, S), where N and Σ are finite, disjoint sets of *nonterminals* and *terminals*, respectively; S in N, is the *start symbol*, and P is a finite list of *productions* of the form $A \rightarrow \alpha$, where A is in N and α in $(N \cup \Sigma)^*$.†

Example: A good example grammar, which will be subsequently referred to as G_0, is $(\{E, T, F\}, \{a, (,), +, *\}, P, E)$, where P consists of the following productions.

(1) $E \rightarrow E+T$
(2) $E \rightarrow T$
(3) $T \rightarrow T*F$
(4) $T \rightarrow F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow a$

G_0 defines arithmetic expressions over operators $+$ and $*$, with no structured variables, and with a representing any identifier.

The way in which a CFG serves to define strings of terminal symbols is as follows.

Definition: Let $G = (N, \Sigma, P, S)$ be a CFG. We define the relation $\underset{G}{\Rightarrow}$ on $(N \cup \Sigma)^*$, by: if α and β are any strings in $(N \cup \Sigma)^*$, and $A \rightarrow \gamma$ is a production, then $\alpha A \beta \underset{G}{\Rightarrow} \alpha\gamma\beta$. If α is in Σ^*, then $\alpha A \beta \underset{Glm}{\Rightarrow} \alpha\gamma\beta$ and if β is in Σ^*, then $\alpha A \beta \underset{Grm}{\Rightarrow} \alpha\gamma\beta$; *lm* and *rm* stand for *leftmost* and *rightmost*, respectively. The *transitive closure* of any relation R, denoted R^+, is defined by:

(1) if aRb, then aR^+b;
(2) if aR^+b, and bR^+c, then aR^+c;
(3) aR^+b only if it so follows from (1) and (2).

The *reflexive-transitive closure* of relation R, denoted R^*, is defined by aR^*b if and only if $a=b$ or aR^*b. Thus, we can use $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$, for example, to express the notion that α can become β by some (possibly null) sequence of replacements of left sides of productions by their right sides.

We will, whenever no ambiguity results, drop the subscript G from the nine relations $\underset{G}{\Rightarrow}$, $\underset{Glm}{\Rightarrow}$, $\underset{Grm}{\Rightarrow}$, and their transitive, and reflexive-transitive closures.

We say $L(G)$, the language *defined by* G, is $\{w \mid w$ is in Σ^* and $S \overset{*}{\Rightarrow} w\}$. $L(G)$ is said to be a *context-free language* (CFL).

Convention: Unless we state otherwise, the following convention regarding symbols of a context free grammar holds.

(1) $A, B, C, \ldots$ are nonterminals.
(2) $\ldots, X, Y, Z$ are either terminals or nonterminals.
(3) $a, b, c, \ldots$ are terminals.
(4) $u, \ldots, z$ are terminal strings.

† If X is any alphabet, then X^* denotes the set of finite length strings of symbols in X, including e, the string of length 0.

(5) $\alpha, \beta, \gamma, \ldots$ are strings of terminals and/or nonterminals.
(6) S is the start symbol; except in G_0, where E is the start symbol.
(7) e denotes the empty string.

We may, using this convention, specify a CFG merely by listing its productions. Moreover, we use the shorthand $A \rightarrow \alpha_1 \mid \cdots \mid \alpha_n$ to stand for the productions $A \rightarrow \alpha_1, A \rightarrow \alpha_2, \ldots, A \rightarrow \alpha_n$.

Definition: Let $G=(N, \Sigma, P, S)$ be a CFG. If $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$, then we say there is a *derivation* of α_n from α_1. If $\alpha_i \underset{lm}{\Rightarrow} \alpha_{i+1}$, $1 \leq i < n$, we call this a *leftmost derivation*, and if $\alpha_i \underset{rm}{\Rightarrow} \alpha_{i+1}$, $1 \leq i < n$, it is a *rightmost derivation*. Most often, we are interested in the case where $\alpha_1 = S$. If $S \overset{*}{\Rightarrow} \alpha$, then α is a *sentential form*. If $S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$, it is a *left sentential form*, and if $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha$, it is a *right sentential form*.

Example: Let us consider G_0, and the string $a*a+a$ in $L(G_0)$. It has the following derivations, among others.

(1) $E \Rightarrow E+T \Rightarrow T+T \Rightarrow T+F$
$\Rightarrow T*F+F \Rightarrow T*a+F \Rightarrow F*a+F$
$\Rightarrow F*a+a \Rightarrow a*a+a$
(2) $E \Rightarrow E+T \Rightarrow T+T \Rightarrow T*F+T$
$\Rightarrow F*F+T \Rightarrow a*F+T \Rightarrow a*a+T$
$\Rightarrow a*a+F \Rightarrow a*a+a$
(3) $E \Rightarrow E+T \Rightarrow E+F \Rightarrow E+a \Rightarrow T+a$
$\Rightarrow T*F+a \Rightarrow T*a+a \Rightarrow F*a+a \Rightarrow a*a+a$

Derivation (1) is neither rightmost nor leftmost. (2) is leftmost and (3) is rightmost. $F*a+F$ is a sentential form of G_0, but happens not to be a left or right sentential form. $a*a+a$ is both a left and right sentential form, and is in $L(G_0)$.

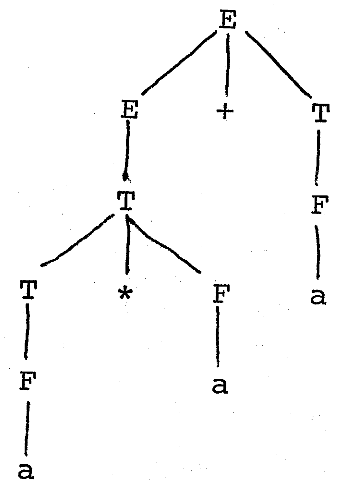
Given a derivation of w from S we can construct a *parse tree* for w as follows. Let $S=\alpha_1 \Rightarrow \alpha_2 \ldots \Rightarrow \alpha_n = w$.

(1) Begin with a single node labeled S. This node is a trivial tree, and is said to be the tree for α_1. In general the tree for α_i will have a leaf *corresponding to* each symbol of α_i. Initially, the lone symbol S of α_1 corresponds to the lone node.
(2) Suppose we have constructed the tree for α_i, $i<n$. Let α_{i+1} be constructed by replacing some instance of A in α_i by β. To this instance of A there corresponds a leaf. Make descendants of that leaf for each symbol of β, ordered from the left. If β is the empty string, then we create a single descendant, labeled e.

The construction of a tree defined above is called *top down*. We can also construct the same tree *bottom up*, as follows.

(1) Begin with an isolated node corresponding to each symbol of α_n. As this algorithm proceeds, we will have a collection of trees corresponding to each step of the derivation. At the end, there will be one tree for the first sentential form α_1 (i.e., S).
(2) Suppose we have a collection of trees for α_i, $1<i \leq n$, where to each symbol of α_i there corresponds a root of one of the trees. If α_i is constructed from α_{i-1} by replacing A by β, create a new node, labeled A, whose direct descendants are the roots of the trees for the symbols of β. The order of symbols in β reflects the order of the descendants. If, however, $\beta=e$, create one descendant for the node labeled A, and label the descendant e.

Example: The unique parse tree for $a*a+a$ in G_0 is shown below. Note that the leaves of the tree read $a*a+a$, from the left.



It is easy to show that the top down and bottom up methods of tree construction yield the same tree when applied to one derivation.

An important property of a CFG is that it be *unambiguous*, that is, no word in its language has more than one parse tree. Because of the way syntax directed translation algorithms work on parse trees, an ambiguous programming language is very likely to provide surprising machine code when (at least some of) its programs are compiled. It is easy to show that the condition of unambiguity is tantamount to saying that each word in the language has a unique leftmost derivation and a unique rightmost derivation.

EARLEY'S ALGORITHM

There is one outstanding method of recognizing the words of an aribtrary CFL and building parse trees, although many others have been proposed. This is the method of Earley.[1] Descriptions of other general methods can be found in References 2 and 3. Earley's method works as follows.

(1) Let $G=(N, \Sigma, P, S)$ and let $a_1 \ldots a_n$ be the word we wish to recognize or parse. We construct lists $I_0, I_1, \ldots, I_n$, of *items*; an item is of the form $[A \rightarrow \alpha \cdot \beta, i]$, where $A \rightarrow \alpha\beta$ is a production and i an integer.

(2) Construct I_0 as follows.
 (i) Add $[S \rightarrow \cdot \alpha, 0]$ to I_0, for each $S \rightarrow \alpha$ in P.
 (ii) If $[A \rightarrow \cdot \mathrm{B}\alpha, 0]$ is on I_0, add $[B \rightarrow \cdot \beta, 0]$ to I_0, if it is not already there, for each $B \rightarrow \beta$ in P.

(3) Suppose we have constructed I_{i-1}. Construct I_i as follows.
 (i) For all $[A \rightarrow \alpha \cdot a_i\beta, j]$ on I_{i-1}, add $[A \rightarrow \alpha a_i \cdot \beta, j]$ to I_i. Recall that a_i is the ith input position.
 (ii) If $[A \rightarrow \alpha \cdot, j]$ is on list I_i, add $[B \rightarrow \beta A \cdot \gamma, k]$ to I_i, if $[B \rightarrow \beta \cdot A\gamma, k]$ is on I_j.
 (iii) If $[A \rightarrow \alpha \cdot B\beta, j]$ is on I_i, add $[B \rightarrow \cdot \gamma, i]$ to I_i for all $B \rightarrow \gamma$ in P.

We can show that item $[A \rightarrow \alpha \cdot \beta, j]$ is placed on I_i if and only if there is a leftmost derivation

$$S \overset{*}{\underset{lm}{\Rightarrow}} a_1 \ldots a_j A\gamma \underset{lm}{\Rightarrow} a_1 \ldots a_j\alpha\beta\gamma \overset{*}{\underset{lm}{\Rightarrow}} a_1 \ldots a_i\beta\gamma.$$

As a special case of the above, $a_1 \ldots a_n$ is in $L(G)$ if and only if $[S \rightarrow \alpha \cdot, 0]$ is on I_n for some α. Thus, it is easy to see how Earley's algorithm can be used to recognize the words of a language. Moreover, once the lists have been constructed, it is possible to build the parse tree of the word. We shall not explore this further here, but the intuitive idea behind the tree construction algorithm is to let the item $[S \rightarrow \alpha \cdot, 0]$ on I_n represent the root, then look for those *complete* (dot at the right end) items which lead to its being placed on I_n. Make these items correspond to the descendants of the root, and proceed top down, finding "causes" for each item corresponding to a node.

Example: The lists for G_0, with input $a*a$ are shown below.

I_0	I_1	I_2	I_3
$E \rightarrow \cdot E+T, 0$	$F \rightarrow a \cdot, 0$	$T \rightarrow T* \cdot F, 0$	$F \rightarrow a \cdot, 2$
$E \rightarrow \cdot T, 0$	$T \rightarrow F \cdot, 0$	$F \rightarrow \cdot (E), 2$	$T \rightarrow T*F \cdot, 0$
$T \rightarrow \cdot T*F, 0$	$E \rightarrow T \cdot, 0$	$F \rightarrow \cdot a, 2$	$E \rightarrow T \cdot, 0$
$T \rightarrow \cdot F, 0$	$T \rightarrow T \cdot *F, 0$		$T \rightarrow T \cdot *F, 0$
$F \rightarrow \cdot (E), 0$			
$F \rightarrow \cdot a, 0$			

List I_3 is constructed as follows. $[F \rightarrow a \cdot, 2]$ is added by rule (3i), because $[F \rightarrow \cdot a, 2]$ is on I_2. $[T \rightarrow T*F \cdot, 0]$ is added to I_3 by rule (3ii), because $[T \rightarrow T* \cdot F, 0]$ is on I_2 and $[F \rightarrow a \cdot, 2]$ is on I_3. $[E \rightarrow T \cdot, 0]$ and $[T \rightarrow T \cdot *F, 0]$ are added because $[E \rightarrow \cdot T, 0]$ and $[T \rightarrow \cdot T*F, 0]$ are on I_0, and $[T \rightarrow T*F \cdot, 0]$ is on I_3. Since $[E \rightarrow T \cdot, 0]$ is on I_3, $a*a$ is in $L(G_0)$.

Earley's algorithm can be implemented in $O(n^2)$ steps of a random access computer if the underlying grammar is unambiguous, and in $O(n^3)$ steps for an arbitrary grammar. Moreover, on many grammars of practical interest, Earley's algorithm operates in $O(n)$ steps. It is the fastest known general parsing algorithm.

LL GRAMMARS

We will now begin the study of several subclasses of the context free grammars. None of the three classes we consider is capable of generating all the context free languages. However, confronted with a context free language that is associated with a real programming language, it is highly likely that grammars in the classes to be discussed do exist.

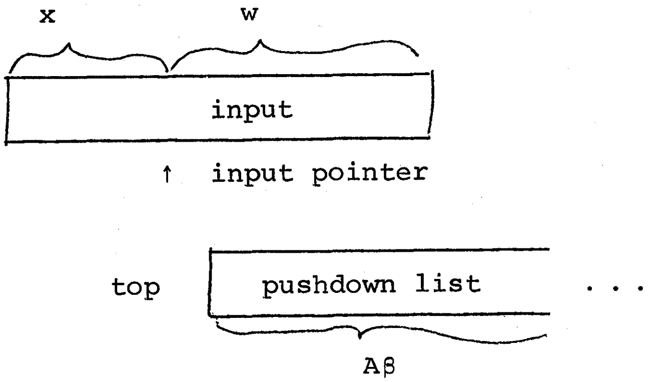
Our first subclass of grammars, called $LL(k)$, for *l*eft to right scan, producing a *l*eftmost derivation, with k symbol lookahead, was first examined in Reference 4. The $LL(l)$ case was developed independently in Reference 5. Development of the theory can be found in References 6 and 7. The general idea can be summarized as follows.

Let $G=(N, \Sigma, P, S)$ be a CFG, and let w be in Σ^*. We may attempt to find a leftmost derivation for w by starting with S and trying to proceed to successive left sentential forms. Suppose we have obtained

$$S=\alpha_1 \underset{lm}{\Rightarrow} \alpha_2 \underset{lm}{\Rightarrow} \cdots \underset{lm}{\Rightarrow} \alpha_i,$$

where $\alpha_i = xA\beta$, and x is a prefix of w. If there are several productions with A on the left, we must select the proper one. It is desirable that we be able to do so using only information which we have accumulated so far during the parse (which we represent by saying that we "know what β is") and the k symbols of w beyond prefix x, for some small k. If we can always make this decision correctly, we say G is $LL(k)$.

If a grammar is $LL(k)$, we can parse it deterministically in a simple fashion. One pushdown list, which holds the portion of a left sentential form to the right of the prefix of terminals ($A\beta$ in the case of α_i, above) is used. (There is also some other information on the list, but we shall not discuss this here. A discussion can be found in in Reference 3.) The input w is scanned left to right, and if α_i has been reached, the input pointer will have scanned over prefix x, and is ready to read the first k symbols of w. The arrangement is shown below.



If production $A \rightarrow \gamma$ is chosen to expand A, the pushdown list is made to hold $\gamma\beta$. Then, any terminals on top of the pushdown list are compared with the input immediately to the right of the input pointer, and the next left sentential form will be properly represented. The process of expanding the topmost nonterminal on the pushdown list then repeats. Note that while expanding nonterminals, we could construct a parse tree top down if we wished.

It should be noted that the pushdown list may carry information other than grammar symbols, in order to summarize at the top of the list, the important information about what is in the list (β in our example). However, this information is not essential, if the grammar is modified properly. We now give the formal definition of an $LL(k)$ grammar.

Definition: If α is a string, let $|\alpha|$ denote the length of α. Let $\alpha{:}k$ denote the first k symbols of α, or all of α if $|\alpha| < k$. Let $G = (N, \Sigma, P, S)$ be a CFG. We say G is $LL(k)$ if whenever we have the following two derivations

(1) $S \underset{lm}{\overset{*}{\Rightarrow}} xA\alpha \underset{lm}{\overset{*}{\Rightarrow}} x\beta\alpha \underset{lm}{\overset{*}{\Rightarrow}} xy$

(2) $S \underset{lm}{\overset{*}{\Rightarrow}} xA\alpha \underset{lm}{\overset{*}{\Rightarrow}} x\gamma\alpha \underset{lm}{\overset{*}{\Rightarrow}} xz$

and $y{:}k = z{:}k$. Then we may conclude that $\beta = \gamma$.

Stated less formally, suppose we are parsing w, and have so far reached left sentential form $xA\alpha$. We have, as indicated in the informal parsing procedure which introduced the section, not scanned w more than k symbols beyond x. Thus, either xy or xz could be w, as far as we know. Suppose $A \rightarrow \beta$ and $\alpha \rightarrow \gamma$ are two productions. Then the $LL(k)$ definition assures us that independent of w, but depending on x, α and the first symbols of w beyond x (which are $y{:}k$, or equivalently, $z{:}k$), we may uniquely select the proper production with which to replace A.

Example: Consider the grammar

$$S \rightarrow aBB \mid b$$

$$B \rightarrow bSS \mid a$$

This grammar is $LL(l)$. For suppose we have two derivations (1) and (2), as in the $LL(k)$ definition. If $y{:}1 = z{:}1 = a$, and A is S, then clearly aBB is both β and γ. If A is B, the $\beta = \gamma = a$. If $y{:}1 = z{:}1 = b$, then $\beta = \gamma = b$ if A is S, and $\beta = \gamma = bSS$ if A is B.

LR GRAMMARS

Just as the $LL(k)$ grammars are a natural class of grammars for which parse trees can be built deterministically, top down, via leftmost derivations, there is a natural class of grammars, called $LR(k)$, for *l*eft to right scan, producing *r*ightmost derivations with k symbol lookahead, for which parse trees can be constructed deterministically bottom up via rightmost derivations. This class of grammars was defined in Reference 8, and the theory, including optimization of LR parsers has been discussed in References 9-13.

Definition: Let

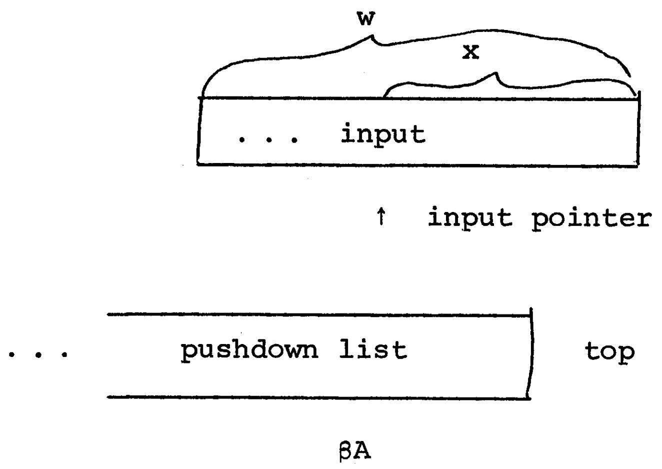
$$S \underset{rm}{\Rightarrow} \alpha Ax \underset{rm}{\Rightarrow} \alpha\beta x.$$

Then β, in the position shown, is said to be a *handle* of right sentential form $\alpha\beta x$. The handle of a right sentential form need not be uniquely defined, but will be, if the grammar is unambiguous.

Given grammar $G = (N, \Sigma, P, S)$ and w in Σ^*, we could attempt to find a rightmost derivation of w, starting with w and working backwards toward S. Suppose we have found

$$S \underset{rm}{\overset{*}{\Rightarrow}} \alpha_i \underset{rm}{\Rightarrow} \alpha_{i+1} \underset{rm}{\Rightarrow} \dots \underset{rm}{\Rightarrow} \alpha_m = w.$$

In order to find the right sentential form previous to α_i, we must find its handle and replace the handle by the left side of the production used to create the handle. If we can do so, we can recognize and parse using a pushdown list as shown.



Suppose α_i is βAx. Then βA will appear on the pushdown list, with A at the top. (As with *LL* grammars, there will be some extra information on the list helping us to make parsing decisions. See References 3 and 8.) x will be a suffix of the input, unscanned to this point. (In the case $i=m$, i.e., $\alpha_i=w$, the pushdown list would be empty, with the input pointer at the left end.) The handle of βAx cannot appear wholly within β, by the definition of a rightmost derivation. Thus, either it is a suffix of βA, or its right end is somewhere within x.

Using the extra information on the pushdown list and the k symbols to the right of the input pointer, the $LR(k)$ parser concludes either

(1) that the handle is not yet on the pushdown list, and an input symbol must be shifted from the input to the pushdown list (i.e., the input pointer moves right, and the symbol which it leaves is placed on top of the pushdown list, or

(2) that the handle is now on top of the pushdown list. In this case, the extra information tells us what production was used at the step which created α_i, and we can *reduce* α_i to α_{i-1} by replacing the handle by the appropriate nonterminal.

Whether (1) or (2) applies, the $LR(k)$ parser repeats the decision whether to shift or reduce. Since we cannot shift indefinitely, we must eventually reduce. If the grammar is $LR(k)$, the correct decision will be made at each step, and we will trace out a rightmost derivation in reverse order. Note that we can build the parse tree bottom up as we do the reductions. We will now give the formal $LR(k)$ definition.

Definition: Let $G=(N, \Sigma, P, S)$ be a CFG, and let $G'=(N\cup\{S'\}, \Sigma, P', S')$ be the *augmented* grammar for G, defined as follows.

(1) S' is a new nonterminal.

(2) P' is P preceded by a production numbered 0, namely $S'\rightarrow S$.

The augmented grammar allows us to treat "reduction" by production $S'\rightarrow S$ as the successful completion of the algorithm.

Suppose we have two derivations in G':

(1) $S'\underset{rm}{\overset{*}{\Rightarrow}}\alpha Ax\underset{rm}{\Rightarrow}\alpha\beta x$

(2) $S'\underset{rm}{\overset{*}{\Rightarrow}}\gamma By\underset{rm}{\Rightarrow}\gamma\delta y$

and we can write $\gamma\delta y$ as $\alpha\beta y'$, where $y'{:}k=x{:}k$. If we can always conclude that $\gamma By=\alpha Ay'$ (i.e., $y=y'$, $A=B$ and $\gamma=\alpha$), then we say G is $LR(k)$.

Stated informally, suppose we are constructing a rightmost derivation of w, in reverse, in such a way that we never observe symbols of w that are more than k symbols beyond the handle. Suppose that we have done some reductions, and reduced some prefix of w to $\alpha\beta$. We still do not know what the tail end of w is; it could be x or y', among other things. However, we do know that the next k symbols of our current right sentential form are $x{:}k$ (equivalently, $y'{:}k$). Then if G is $LR(k)$, we may determine that β is the handle and must be reduced to A. We can make this determination independent of whether the input actually ends with x or y'.

Example: Consider the grammar

$$S\rightarrow aS \mid a$$

The grammar is not $LR(O)$. For in the augmented grammar, we have derivations

$$S'\underset{rm}{\overset{*}{\Rightarrow}}S\underset{rm}{\Rightarrow}a$$

$$S'\underset{rm}{\overset{*}{\Rightarrow}}aS\underset{rm}{\Rightarrow}aa$$

We may compare these derivations with those of the $LR(k)$ definition, letting $\alpha=e$, $A=B=S$, $\beta=\gamma=\delta=a$, $x=y=e$ and $y'=a$. Then $x{:}0=y'{:}0=e$, but γBy, which is aS, is not equal to $\alpha Ay'$, which is Sa.

However, this grammar is $LR(l)$. The only possible right sentential forms $\alpha\beta x$ are:

(i) $\alpha=x=e$, $\beta=S$, in which the last step replaced S' by S,

(ii) $\alpha=a^i$ for some $i\geq 0$, $x=e$, and $\beta=aS$. (The last step replaced S by aS.), or

(iii) $\alpha=a^i$, $\beta=a$ and $x=e$, where S was replaced by a at the last step.

The above three possibilities hold also if we refer to γ, δ and y instead of α, β and x. If we have a violation of the $LR(1)$ condition, since $x{:}1=e$, we must have $\gamma\delta y=\alpha\beta y'$ and $y'{:}1=e$, but $\gamma By\neq\alpha Ay'$. If $y'{:}1=e$, then $y'=e$. Moreover, as we argued in (i)-(iii) above, we must have $y=e$ in the right sentential form $\gamma\delta y$. Thus, $\gamma\delta=\alpha\beta$.

If $\alpha\beta=S$, then $\gamma\delta=S$, and $\gamma=e$ and $\delta=S$ follows. Then $B=A=S'$, and we have $\gamma By=\alpha Ay'$, which we assumed not to be the case.

If $\alpha=a^i$ and $\beta=aS$, it similarly follows that $\gamma=a^i$ and $\delta=aS$; $A=B=S$. Again $\gamma By=\alpha Ay'$.

Finally, if $\alpha=a^i$ and $\beta=a$, we may conclude that $\gamma=a^i$ and $\delta=a$. Again, $A=B=S$, and $\gamma By=\alpha Ay'$.

We conclude that the grammar is $LR(l)$.

PRECEDENCE GRAMMARS

Many practical classes of grammars admit of shift-reduce type parsing algorithms, as outlined in the previous section, but without the need for extra information on the pushdown list. We shall here discuss several classes that go under the heading of precedence grammars. The initial idea of operator precedence parsing is from Reference 14, while the most common current notion of a precedence grammar first appeared in Reference 15. The theory of precedence parsing has been developed in References 16 through 21. We now give the basic precedence definitions.

Definition: Let $G=(N, \Sigma, P, S)$ be a CFG with no production $A\rightarrow e$, no derivation $A\overset{+}{\Rightarrow}A$, and no symbols not appearing in the derivation of some word in $L(G)$. We assume $ is not in $N\cup\Sigma$. We define three *precedence relations*, $<\cdot$, $\doteq$ and $\cdot>$ on $N\cup\Sigma$ as follows.

(1) $X\doteq Y$ if and only if there is a production of the form $A\rightarrow\alpha XY\beta$.
(2) $X<\cdot Y$ if and only if there is a production $A\rightarrow\alpha XB\beta$ and $B\overset{+}{\Rightarrow}Y\gamma$.
(3) $X\cdot>Y$ if and only if Y is a terminal, there is a production $A\rightarrow\alpha BZ\beta$, $B\overset{+}{\Rightarrow}\gamma X$ and $Z\overset{*}{\Rightarrow}Y\delta$.

We also have $\$<\cdot X$ if $S\overset{+}{\Rightarrow}X\alpha$ and $X\cdot>\$$ if $S\overset{+}{\Rightarrow}\beta X$.

A grammar is *uniquely invertible* if it has no two productions of the form $A\rightarrow\alpha$ and $B\rightarrow\alpha$.

We say G is a *precedence grammar* if it satisfies the constraints mentioned above (no production $A\rightarrow e$, etc.), and the relations $<\cdot$, $\doteq$ and $\cdot>$ are disjoint from one another.

If, in addition, G is uniquely invertible, we say G is a *simple precedence grammar*.

Example: Let us consider the grammar with productions $S\rightarrow AA$, $A\rightarrow aA \mid b$. It is not a precedence grammar, because we have $A\cdot>a$ and $A<\cdot a$. That is, since AA is a right side, $A\overset{+}{\Rightarrow}aA$ and $A\overset{*}{\Rightarrow}aA$, we have $A\cdot>a$. Since AA is a right side and $A\overset{+}{\Rightarrow}aA$ again, we conclude $A<\cdot a$. However, the modified grammar

$$S\rightarrow BA$$

$$B\rightarrow A$$

$$A\rightarrow aA \mid b$$

is a simple precedence grammar. Its precedence relations are shown below.

	S	B	A	a	b	$
S						
B			≐	<·	<·	
A				·>	·>	·>
a			≐	<·	<·	
b				·>	·>	·>
$		<·	<·	<·	<·	

We can parse according to a simple precedence grammar as follows. An input, scanned left to right by a pointer is used, as is a pushdown list. Right sentential forms are represented as in the description of the $LR(k)$ parser. Initially, only $ appears on the pushdown list, and the leftmost symbol of the input is being scanned. The input has $ appended at the right. At each step of the algorithm, the following is done. Let $X_1\ldots X_m$ appear on the pushdown list (top right) and $a_1\ldots a_r$ be the remaining input.

(1) If $X_m<\cdot a_1$ or $X_m\doteq a_1$, a_1 is shifted onto the pushdown list, and the process repeats.
(2) If $X_m\cdot>a_1$, find k such that $X_{k-1}<\cdot X_k\doteq X_{k+1}\doteq\ldots\doteq X_m$. Let $A\rightarrow X_k\ldots X_m$ be a production. (A must be unique if G is a simple precedence grammar.) Then replace $X_k\ldots X_m$ by A on top of the pushdown list.
(3) If neither (1) nor (2) apply, accept the input if $X_l\ldots X_m=\$S$ and $a_l\ldots a_r=\$$. Declare an error otherwise.

SYNTAX DIRECTED TRANSLATIONS

We now turn to formalisms for specifying the code generation phase of a compiler. The general strategy is to work from a parse tree, building some "translations" at each node. One translation at the root of the tree is designated the output of the translation system. The most elementary system, called a (formal) *syntax directed translation scheme* was first expounded in Reference 22 and formalized in Reference 4. Some theoretical developments and generalizations appear in References 22 through 30.

Definition: A (formal) *syntax directed translation* scheme (SDTS) is a CFG $G=(N, \Sigma, P, S)$, together with a translation element for each production. The *translation element* associated with production $A \rightarrow \alpha$ is a string β in $(N \cup \Delta)^*$, where Δ is an alphabet of *output symbols*, disjoint from N. β must be such that there is a one to one mapping from its nonterminals to the nonterminals of α, where each nonterminal is mapped to an identical symbol. If the mapping preserves the (left to right) order of appearance of the nonterminals in α and β, then the SDTS is said to be *simple*.

We construct a translation from a parse tree in the CFG as follows. An order for the interior nodes is chosen so that each node follows all its descendants in the order. Each node is considered in its turn. If a particular node n has label A and its direct descendants have labels $X_1, \ldots, X_n$, from the left, then

$$A \rightarrow X_1 \ldots X_n$$

is a production. Let $\beta = Y_1 \ldots Y_m$ be the translation element for that production. The translation at n is formed from $Y_1 \ldots Y_m$ by substituting for each Y_i in N, the translation at the jth direct descendant of n if Y_i is associated with X_j.

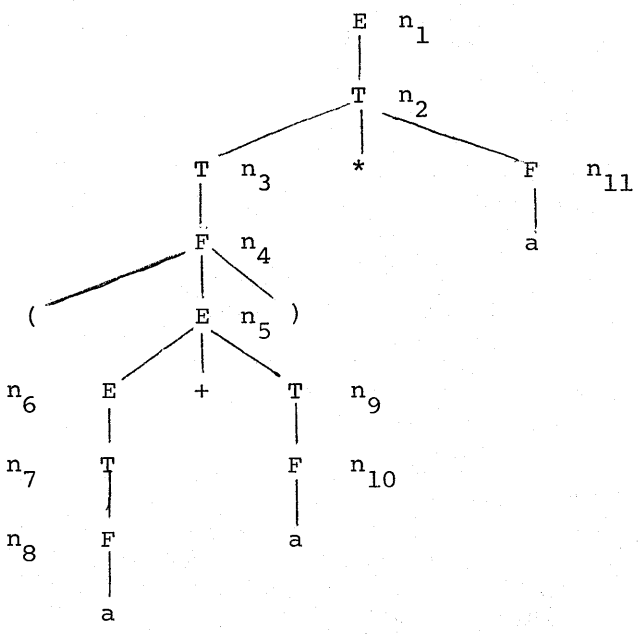
The *translation* defined by the SDTS is the set of pairs (w, x) such that w is the yield of some parse tree T and the translation defined at the root of T is x.

Example: We can build a translation on G_0 to translate infix arithmetic expressions to postfix (operator follows both operands) expressions.

production	*Translation element*
$E \rightarrow E+T$	$ET+$
$E \rightarrow T$	T
$T \rightarrow T*F$	$TF*$
$T \rightarrow F$	F
$F \rightarrow (E)$	E
$F \rightarrow a$	a

Since no production of G_0 has more than one occurrence of the same nonterminal on the right side of any production, the mapping between nonterminals of the translation elements and their productions is obvious. Note that the SDTS is simple.

The string $(a+a)*a$ has the parse tree shown below. Nodes have been numbered for reference. The following order of the interior nodes is acceptable. $n_8, n_7, n_6, n_{10}, n_9, n_5, n_4, n_{11}, n_2, n_1$.



The translation at n_8 is just a, since production $F \rightarrow a$ was applied there, and the associated translation element is a. To compute the translation at n_7, we substitute the translation at n_8 for F in the translation element associated with $T \rightarrow F$. Thus, the translation at n_7 is a, as is the translation at n_6, n_{10}, and n_9. The translation at n_5 is computed by substituting the translation at n_6 for E and that at n_9 for T in the translation element $ET+$. Thus, the translation at n_5 is $aa+$. Proceeding similarly, the translation at n, is $aa+a*$.

REFERENCES

1 J EARLEY
An efficient context free parsing algorithm
CACM 13 pp 94-102 February 1970

2 T E CHEATHAM
The theory and construction of compilers
Unpublished notes Computer Associates Wakefield Mass 1967

3 A V AHO J D ULLMAN
The theory of parsing, translation and compiling
Prentice-Hall Englewood Cliffs NJ to appear October 1972

4 P M LEWIS R E STEARNS
Syntax-directed transduction
JACM 15 pp 464-488 July 1968

5 D E KNUTH
Top down syntax analysis
Intl Summer School on Computer Programming Copenhagen Denmark 1967

6 D J ROSENKRANTZ R E STEARNS
Properties of deterministic top-down grammars
Inf and Control 17 pp 226-256 1970

7 D WOOD
The theory of left factored languages
Part I—Computer J 12 pp 349-356 December 1969 Part II—13 pp 55-62 February 1970

8 D E KNUTH
On the translation of languages from left to right
Inf Control 8 pp 607-639 October 1965

9 A J KORENJAK
A practical method for constructing LR(k) processors
CACM 12 pp 613-623 1969

10 F L DE REMER
Practical translators for LR(k) languages
MIT PhD Thesis Cambridge Mass 1969

11 F L DE REMER
Simple LR(k) grammars
CACM 14 pp 453-459 July 1971

12 A V AHO J D ULLMAN
The care and feeding of LR(K) grammars
Proc 3rd Annual ACM Symposium on Theory of Computing pp 159-170 1971

13 A V AHO J D ULLMAN
A technique for speeding up LR(k) parsers
Unpublished memorandum Bell Telephone Laboratories Murray Hill NJ 1972

14 R W FLOYD
Syntactic analysis and operator precedence
JACM 10 pp 316-333 July 1963

15 N WIRTH H WEBER
EULER—A generalization of ALGOL, and its formal definition, Part I
CACM 9 pp 13-25 January 1966

16 W M McKEEMAN
An approach to computer language design
Tech Report CS48 Computer Science Dept Stanford University Stanford California

17 M J FISCHER
Some properties of precedence languages
Proc ACM Symp on Theory of Computing pp 181-190 May 1969

18 S L GRAHAM
Extended precedence languages, bounded right context languages and deterministic languages
Proc IEEE 11th Annual Symp on Switching and Automata Theory pp 175-180 October 1970

19 J N GRAY
Precedence parsers for programming languages
PhD Thesis Dept of Computer Science University of California Berkeley California September 1969

20 A V AHO P J DENNING J D ULLMAN
Weak and mixed strategy precedence parsing
JACM to appear April 1972

21 A COLMERAUER
Total precedence relations
JACM 17 pp 14-30 January 1970

22 E T IRONS
A syntax directed compiler for ALGOL 60
CACM 4 pp 51-55 January 1961

23 A V AHO J D ULLMAN
Properties of syntax directed translations
J Computer and System Sciences 3 pp 319-334 August 1969

24 A V AHO J D ULLMAN
Syntax directed translations and the pushdown assembler
J Computer and System Sciences 3 pp 37-56 February 1969

25 L PETRONE
Syntax directed mappings of context free languages
Conference Record of 9th Annual Symposium on Switching and Automata Theory pp 160-175 October 1968

26 A V AHO J D ULLMAN
Translations on a context-free grammar
Proc ACM Symposium on Theory of Computing pp 93-112 May 1969

27 W C ROUNDS
Mappings and grammars on trees
Mathematical Systems Theory 4 pp 257-287 July 1970

28 J W THATCHER
There's a lot more to finite automata theory than you would have thought
Proc 4th Annual Princeton Conference on Information Sciences and Systems Princeton NJ March 1970

29 D E KNUTH
Semantics of context free languages
Math Systems Theory 2 pp 127-146 June 1968

30 T WILCOX
Generating machine code for high level programming languages
Cornell University Dept of Computer Science TR-71-103 September 1971

The Terminal IMP for the ARPA computer network*

by S. M. ORNSTEIN, F. E. HEART, W. R. CROWTHER, H. K. RISING, S. B. RUSSELL and A. MICHEL

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

A little over three years ago the Advanced Research Projects Agency of the Department of Defense (ARPA) began implementation of an entirely new venture in computer communications: a network that would allow for the interconnection, via common-carrier circuits, of dissimilar computers at widely separated, ARPA-sponsored research centers. This network, which has come to be known as the ARPA Network, presently includes approximately 20 nodes and is steadily growing. Major goals of the network are (1) to permit resource sharing, whereby persons and programs at one research center may access data and interactively use programs that exist and run in other computers of the network, (2) to develop highly reliable and economic digital communications, and (3) to permit broad access to unique and powerful facilities which may be economically feasible only when widely shared.

The ARPA Network is a new kind of digital communication system employing wideband leased lines and message switching, wherein a path is not established in advance and instead each message carries an address. Messages normally traverse several nodes in going from source to destination, and the network is a store-and-forward system wherein, at each node, a copy of the message is stored until it is safely received at the following node. At each node a small processor (an *Interface Message Processor*, or *IMP*) acts as a nodal switching unit and also interconnects the research computer centers, or *Hosts*, with the high bandwidth leased lines.

A set of papers presented at the 1970 SJCC[1–5] described early work on the ARPA Network in some detail, and acquaintance with this background material (especially Reference 2) is important in understanding the current work. The present paper first discusses major developments that have taken place in the network over the last two years. We then describe the *Terminal IMP*, or *TIP*, a development which permits direct terminal access to the network. Finally we mention some general issues and discuss plans for the next stages in development of the network.

THE DEVELOPING NETWORK

The initial installation of the ARPA Network, in 1969, consisted of four nodes in the western part of the United States. A geographic map of the present ARPA Network is shown in Figure 1. Clearly, the most obvious development has been a substantial *growth*, which has transformed the initial limited experiment into a national assemblage of computer resources and user communities. The network has engendered considerable enthusiasm on the part of the participants, and it is increasingly apparent that the network represents a major new direction in both computer and communications technology.

Figure 2 is a logical map of the network, where the Host computer facilities are shown in ovals, all circuits are 50 kilobits, and dotted circuits/nodes represent planned installations. On this figure certain nodes are listed as a "316 IMP"; this machine is logically nearly identical to the original IMP, but can handle approximately two-thirds of the communication traffic bandwidth at a cost savings of approximately one-half. The original IMP includes a Honeywell 516 computer, and more recently Honeywell began to market the 316 computer as a cheaper, downward-compatible machine. As the network has grown, sites were identified which did not require the full bandwidth of the original IMP, and a decision was made to provide an IMP version built around the 316 computer. Also shown in Figure 2 are certain nodes listed as "TIP"; this new machine

* This work was sponsored by the Advanced Research Projects Agency under Contract No. DAHC15-69-C-0179.

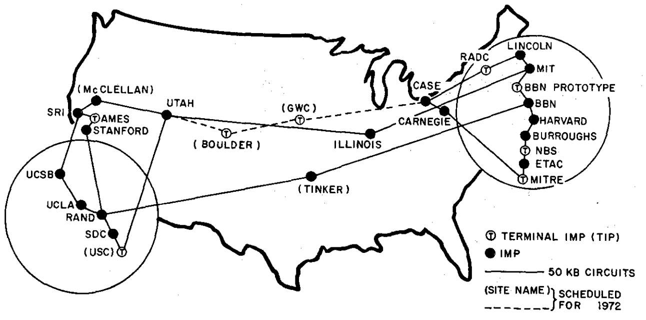


Figure 1—ARPA Network, geographic map, December 1971

is discussed in detail later in this paper. Site abbreviations shown on Figures 1 and 2 are explained in Table I.

As the network has grown, a great deal of work has been concentrated on the development of Host-to-Host protocol procedures. In order for programs within one Host computer system to communicate with programs in other Hosts, agreed-upon procedures and formats must be established throughout the network. This problem has, as predicted, turned out to be a difficult one. Nonetheless protocol procedures have evolved and are being accepted and implemented throughout the net. At the present writing, many of the Hosts have working "network control programs" which implement this protocol. Protocol development is more fully reported in a companion paper,[6] but we wish to make a general observation on this subject: the growth of the network has dynamically catalyzed an area of computer science which has to do with the quite general problem of *how programs should intercommunicate,* whether in a single computer or between computers. Thus the evolution of the Host-to-Host protocol represents a side benefit of the network that reaches well beyond its utility to the network alone.

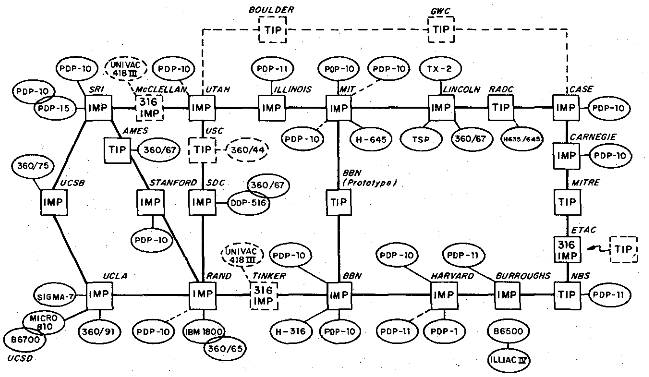


Figure 2—ARPA Network, logical map, December 1971

Since both hardware and software network connections must be implemented by each Host, it is important that the external characteristics of the IMP be relatively stable. This stability has been carefully maintained, while at the same time internal operation of the IMP program has undergone extensive revision and improvement. For example, trouble reporting, statistics gathering, and test procedures have been substantially improved. In addition to improvements that have already been incorporated into the program, there have also been extensive studies of performance and message flow control.[7] These studies have pointed up areas of vulnerability to perverse heavy traffic patterns and have suggested still other possible improvements in the routing and flow control mechanisms. Potential changes are presently being being studied in some detail and will be incorporated into one or more forthcoming major revisions of the program. They will hopefully anticipate problems which might be expected to arise as traffic flow in the network becomes heavier.

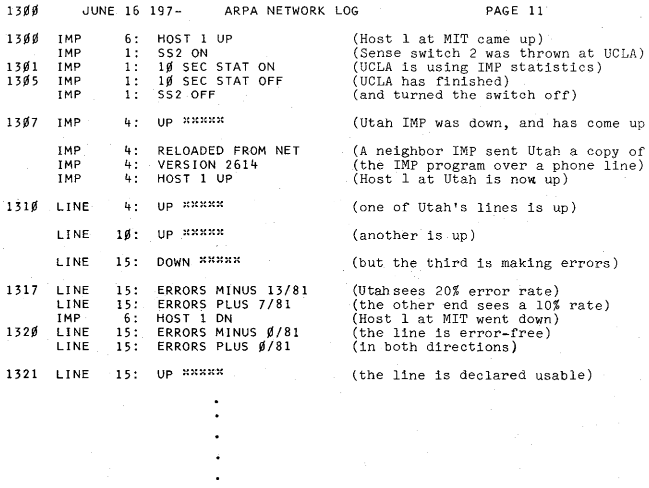
TABLE I—Site Abbreviations

NCAR	National Center for Atmospheric Research
GWC	Global Weather Central
SRI	Stanford Research Institute
MC CLELLAN	McClellan Air Force Base
UTAH	University of Utah
ILLINOIS	University of Illinois
MIT	Massachusetts Institute of Technology
LINCOLN	M.I.T. Lincoln Laboratory
RADC	Rome Air Development Center
CASE	Case Western Reserve
AMES	N.A.S.A. Ames Research Center
USC	University of Southern California
UCSB	University of California at Santa Barbara
STANFORD	Stanford University
SDC	Systems Development Corporation
BBN	Bolt Beranek and Newman
CARNEGIE	Carnegie University
MITRE	MITRE Corporation
ETAC	Environmental Technical Applications Center
UCLA	University of California at Los Angeles
RAND	Rand Corporation
TINKER	Tinker Air Force Base
HARVARD	Harvard University
BURROUGHS	Burroughs Corporation
NBS	National Bureau of Standards

Somewhat belatedly in the network design, the need to connect a single IMP to several Hosts was recognized. This required multiple Host interfaces at the IMP as well as more complex IMP software. Further, the various Host computers at a site are often physically distant from one another, thus requiring an increase in the maximum allowable physical separation between a Host and its IMP. To connect to an arbitrarily remote Host would have meant a communications interface capable of attachment to common-carrier circuits via modems. It would furthermore have required cooperative error control from the Host end. At the time, we chose not to modify the logical way in which the IMP dealt with the Host and instead we provided more sophisticated line drivers which would handle distances of up to two thousand feet. Several such "Distant Host" installations are now working in the network. Unfortunately, as the network has grown, new sites have appeared where still greater Host/IMP distances are involved. The present scheme does not include error control, and use of this scheme over greater distances is not appropriate. At the present time we are therefore considering how best to arrange IMP/Host connections over large distances and additional options will be required in this domain.

Another facility which has been tested is the ability of the IMPs to communicate over 230.4 kilobit phone lines instead of 50 kilobit lines. A short, fast link was incorporated into the network for a brief period and no problems were encountered. To date, network loading has not justified upgrading any operational network circuits to 230.4 kilobits, but this will be considered as loading rises.

Substantial effort has gone into traffic and trouble reporting. A Network Control Center (NCC) has been built at Bolt Beranek and Newman Inc. in Cambridge, where a small dedicated Host computer receives reports each minute from every IMP on the network. Traffic summaries and status and trouble reports are then generated from this material. Specifically, a logger records any changes that the IMPs report to the NCC; it records line errors and IMP storage counts when they exceed certain limits, as well as unusual events, such as reloading from the net, checksum errors, etc. Figure 3 shows an example of this log. (The comments, in parentheses to the right, are not part of the log but have been added to explain the meaning of the entries.) In addition to this detailed log of interesting events, one may at any time obtain a quick summary of the status of the network. Finally, detailed and summary logs of Host and line traffic are produced. The NCC is a focal point not only for monitoring the network but also for testing and diagnosing remote troubles. Lines throughout the network can be looped from here in order to isolate difficulties. Personnel of the center coordinate all debugging, maintenance, repair and modification of equipment.



```
13ØØ     JUNE 16 197-     ARPA NETWORK LOG                 PAGE 11

13ØØ  IMP      6:  HOST 1 UP                (Host 1 at MIT came up)
      IMP      1:  SS2 ON                   (Sense switch 2 was thrown at UCLA)
13Ø1  IMP      1:  1Ø SEC STAT ON           (UCLA is using IMP statistics)
13Ø5  IMP      1:  1Ø SEC STAT OFF          (UCLA has finished)
      IMP      1:  SS2 OFF                  (and turned the switch off)

13Ø7  IMP      4:  UP *****                 (Utah IMP was down, and has come up

      IMP      4:  RELOADED FROM NET        (A neighbor IMP sent Utah a copy of
      IMP      4:  VERSION 2614             (the IMP program over a phone line)
      IMP      4:  HOST 1 UP                (Host 1 at Utah is now up)

131Ø  LINE     4:  UP *****                 (one of Utah's lines is up)

      LINE    1Ø:  UP *****                 (another is up)

      LINE    15:  DOWN *****               (but the third is making errors)

1317  LINE    15:  ERRORS MINUS 13/81       (Utah sees 20% error rate)
      LINE    15:  ERRORS PLUS 7/81         (the other end sees a 10% rate)
      IMP      6:  HOST 1 DN                (Host 1 at MIT went down)
132Ø  LINE    15:  ERRORS MINUS Ø/81        (the line is error-free)
      LINE    15:  ERRORS PLUS Ø/81         (in both directions)

1321  LINE    15:  UP *****                 (the line is declared usable)

                     .
                     .
                     .
                     .
                     .
```

Figure 3—Typical segment of NCC log

DIRECT TERMINAL ACCESS

During the early phases of network development a typical node has consisted of one or more large time-shared computer systems connected to an IMP. The IMPs at the various sites are connected together into a subnet by 50 kilobit phone lines and the large Host computers communicate with one another through this subnet. This arrangement provides a means for sharing resources between such interconnected centers, each site potentially acting both as a user and as a provider of resources. *This total complex of facilities constitutes a nationwide resource which could be made available to users who have no special facilities of their own to contribute to the resource pool.* Such a user might be at a site either with no Host computer or where the existing computer might not be a terminal-oriented time-sharing system.

A great deal of thought went into considering how best to provide for direct terminal access to the network. One possibility, which would have essentially been a non-solution, was to require a user to dial direct to the appropriate Host. Once connected he could, of course, take advantage of the fact that that Host was tied to other Hosts in the net; however, the network lines would not have been used to facilitate his initial connection, and such an arrangement limits the terminal bandwidth to what may be available on the switched common-carrier networks.

A similar solution was to allow terminals to access the network through a Host at a nearby node. In such a case, for example, a worker in the New England area wishing to use facilities at a California site might connect into a local Boston Host and use that Host as a tap into the network to get at the facilities in California. This approach would have required Hosts to provide hardware access facilities for many terminals which would use their systems only in passing. For many Hosts, the kinds of terminals which can be connected directly are limited in speed, character set, etc. In terms of reliability, the user would have been dependent on the local Host for access; when that Host was down, his port into the network would be unavailable. Furthermore, the Hosts would have been confronted with all of the problems of composing terminal characters into network messages and vice versa as well as directing these messages to the proper terminals and remote Hosts. Time-sharing systems are generally already overburdened with processing of characters to and from terminals and many are configured with front end processors employed explicitly to off-load this burden from the main processor. Increasing the amount of such work for the Hosts therefore seemed unreasonable and would have resulted in limiting terminal access. Instead, a completely separate means for accessing the network directly seemed called for: an IMP with a flexible terminal handling capability—a *Terminal IMP*, or *TIP*.

One of the fundamental questions that arises when considering this problem is: what is the proper distribution of computational power among the remote big facility, the terminal processor, and the terminal? Shall the terminal processor be clever, have sizable storage, be user programmable, etc., or shall it be a simpler device whose basic job is multiplexing in a flexible way? Serious work with interactive graphics seems to require the terminal to include, or be in propinquity to, a user-programmable processor and considerable storage. To date, such work has primarily been done with terminals attached directly to a Host. Some elaborate terminals, such as the Adage, include a powerful processor. Other kinds of terminals, including Teletype-like devices, alphanumeric displays, or simple graphic displays (excluding serious interactive graphics), do *not* require a user-programmable local processor or significant local storage.

We believe that the great majority of potential groups needing access to the ARPA Network will not need powerful interactive graphics facilities. Further, for the minority that will need powerful terminals, we believe that individual terminals with built-in or accompanying processors will become more common. For these reasons, we decided that the terminal processor should be simple and not programmable from the terminals. *The computational load and the storage should be in the Hosts or in the terminals and not in the terminal processor.* This simple multiplexing approach is amenable to some standardization and is philosophically close to the original IMP notion of a standard nodal device.

Another major question we faced was whether to build a separate terminal handler and then connect it to the IMP or to build an integrated unit that was housed in a common cabinet and used the IMP processor. One advantage of the two-machine approach is that it isolates the IMP functions from the terminal functions, thereby providing a barrier of safety for the net. This approach also provides the processing power of two machines and a potentially greater degree of user freedom in modifying or writing programs. Another interesting reason for considering separate machines was to reduce the large cost associated with I/O equipment (such as line controllers) by making use of the extra processing power. We discussed with several manufacturers the possibility of bringing terminal wires "into" their processors and decoding the basic line information directly in software. However, even with some of the new state-of-the-art machines, like the Meta-4, with fast 90 nsec read-only memories to handle character decoding, the I/O cost was still high, and in large part the necessary I/O equipment was yet to be designed. We therefore concluded that it was still somewhat early to proceed in this fashion, and two processors did not appear to save I/O equipment.

The principal disadvantages of the two-machine approach were the higher initial cost, the difficulties of maintaining two machines, and the software problems of dealing with two machines. In particular, the communication between two machines would require two hardware Host interfaces, and two software Host/IMP programs. This would result in a much poorer communication between the IMP program and the terminal handling program than would exist in a single machine. In either case, one machine or two, the

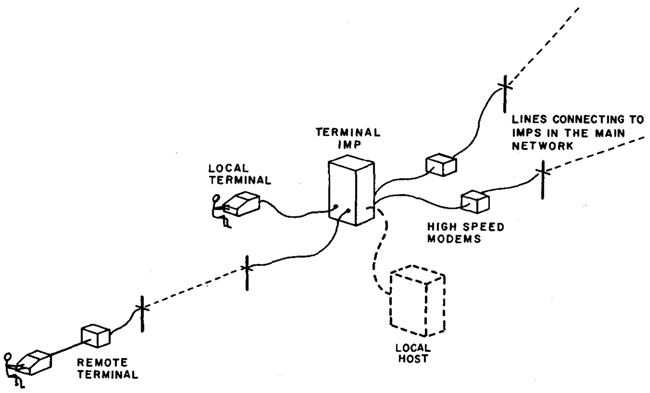


Figure 4—A TIP in the network

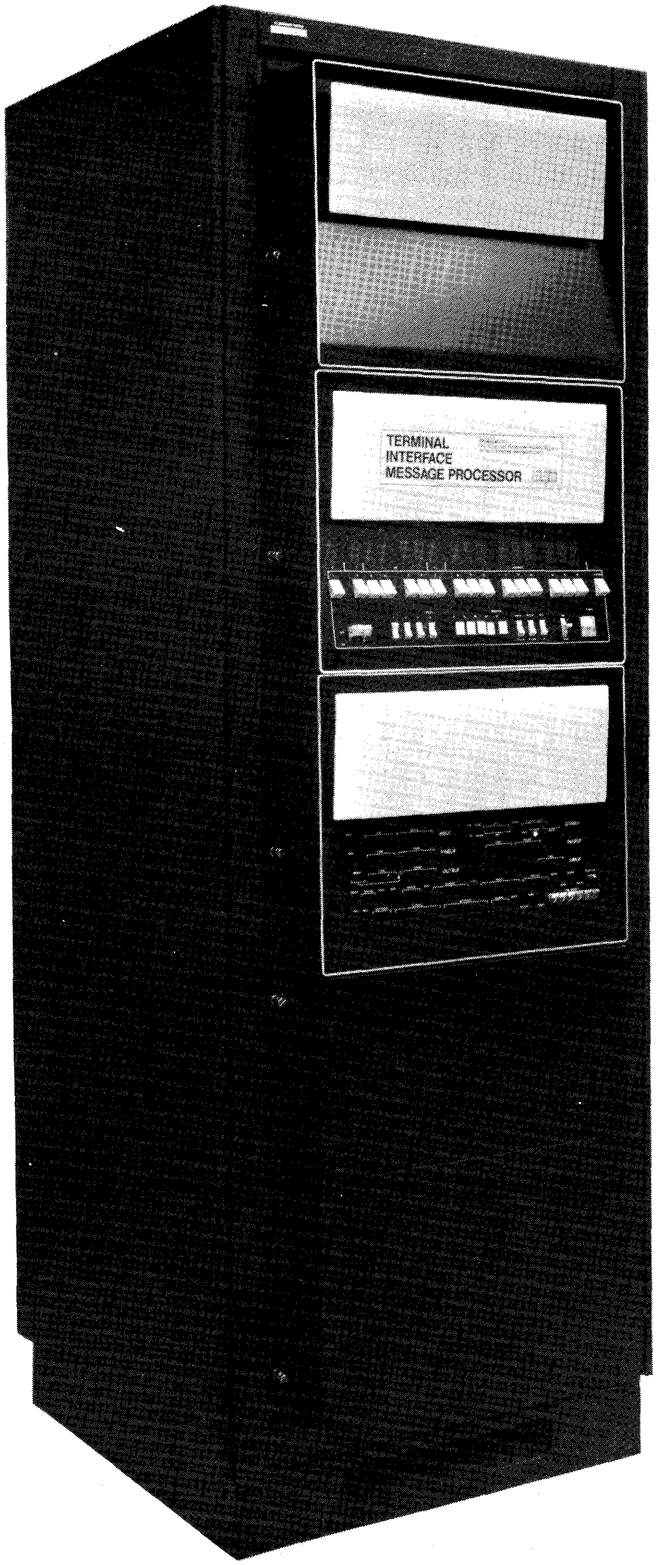


Figure 5—Photograph of TIP

terminal handling process required the implementation of a version of Host protocol.

We finally decided that the least expensive and most sensible technical alternative was to build the IMP and the terminal handler (with the necessary subset of Host protocol) together into a single machine. Then, since certain new machines appeared attractive in cost/performance relative to the Honeywell 16 series, we spent considerable time thinking about what machine to use. We decided that no alternate choice was sufficiently attractive to justify rewriting the main IMP program and redesigning the Host and modem interface hardware. This left us with a decision between the Honeywell 516 and 316 computers. We chose the 316 on the basis of size and cost, feeling that the somewhat higher bandwidth of the 516 was not essential to the job and did not justify its higher price and the second cabinet which would have been required.

TERMINAL IMP HARDWARE

Figure 4 shows how the TIP fits the user into the network. Up to 63* terminals, either remote or local, of widely diverse characteristics may be connected to a given TIP and thereby "talk" into the network. It is also possible to connect a Host to a TIP in the usual way.

The TIP is shown in Figure 5. It is built around a Honeywell H-316 computer with 20K (20,480 words) of core. It embodies a standard 16 port multiplexed memory channel with priority interrupts and includes a Teletype for debugging and program reloading. Other features of the standard IMP also present are a real-time clock, power-fail and auto-restart mechanisms, and a program-generated interrupt feature.[2] As in the standard IMP, interfaces are provided for connecting to high-speed (50 kilobit, 230.4 kilobit, etc.) modems as well as to Hosts. The single-cabinet version limits the configuration to a total of three modem and/or Host interfaces, but an expansion cabinet may be used to increase this limit. More basic limits are set by the machine's logical organization (specifically the number of available memory channels) and the program bandwidth capability as discussed below.

Aside from the additional 8K of core memory, the primary hardware feature which distinguishes the TIP from a standard IMP is a Multi-Line Controller (MLC) which allows for connection of terminals to the

* There are 64 hardware lines but line Ø is logically reserved by the program for special use.

IMP. Any of the MLC lines may go to local terminals or via modems to remote terminals. As shown in Figure 6 the MLC consists of two portions, one a piece of central logic which handles the assembly and disassembly of characters and transfers them to and from memory, and the other a set of individual Line Interface Units (all identical except for small number of option jumpers) which synchronize reporting to individual data bits between the central logic and the terminal devices and provide for control and status information to and from the modem or device. Line Interface Units may be physically incorporated one at a time as required.

The MLC connects to the high-speed multiplexed memory channel option of the H-316, and uses three of its channels as well as two priority interrupts and a small number of control instructions. The MLC is fabricated by BBN and is built from TTL integrated circuits. The MLC central controller, complete with power supply, is housed in an H-316 expansion chassis, and the entire MLC, as well as the computer itself, is mounted in a standard six-foot rack. Additional space is provided in the bottom of the rack for up to sixteen card-mounted modems of the Bell 103, 201, or 202 variety, together with their power supplies.

In order to accommodate a variety of devices, the controller handles a wide range of data rates and character sizes, in both synchronous and asynchronous modes. Data characters of length 5-, 6-, 7-, or 8-bits are allowed by the controller. Since no interpretation of characters is done by the hardware, any character set, such as Baudot, Transcode ASCII or EBCDIC may be used.

The following is a list of data rates accepted by the controller:

SYNCHRONOUS	*ASYNCHRONOUS* (Nominal Rates)		
Any rate up to and including 19.2 Kb/s	75	1200	
	110	1800	
	134.5	2400	
	150	4800	output only
	300	9600	output only
	600	19200	output only

All above in bits/second

The data format required of all devices is bit serial and each character indicates its own beginning by means of a start bit which precedes the data and includes one or more stop bits at the end of the character. This per-character framing is quite standard for asynchronous lines but synchronous lines, generally designed for higher bandwidths, frequently adopt some form of "binary synchronous communication" where the characters are packed tightly together into messages which are then framed by special characters. Framing is thereby amortized over the entire message, thus consuming a smaller fraction of the available bandwidth than the per-character framing which uses two or more bits for every character. The difficulties with this scheme, however, are that it is more complex, requiring more sophisticated hardware at each end, and that no real standards exist which are adhered to by all or even most types of synchronous devices. We therefore decided to adopt per-character framing with start and stop bits even on synchronous lines. At a cost of some twenty percent of the bandwidth for framing, a very simple and general scheme is thus arrived at. A number of high speed terminal manufacturers, faced with the same problems, have arrived at a similar conclusion.

Given these characteristics, then, the controller will connect to the great majority of normal terminal devices such as Teletypes, alphanumeric CRT units, and modems, and also (with suitable remote interface units) to many peripheral devices such as card readers, line printers, and graphics terminals. Either full or half duplex devices can be accommodated. The standard TIP program cannot deal with a magnetic tape unit through the MLC. However, as a special option, and with the use of additional core memory, standard Honeywell tape drives can be connected to the TIP as normal peripherals.

The individual terminal line levels are consistent with EIA RS-232C convention. Data rates and character length are individually set for each line *by the the program*. For incoming asynchronous lines, the program includes the capability for detecting character length and line data rate as discussed below.

Logically, the controller consists of 64 input ports and 64 output ports. Each input/output pair is brought out to a single connector which is normally connected to a single device. However, by using a special "Y"

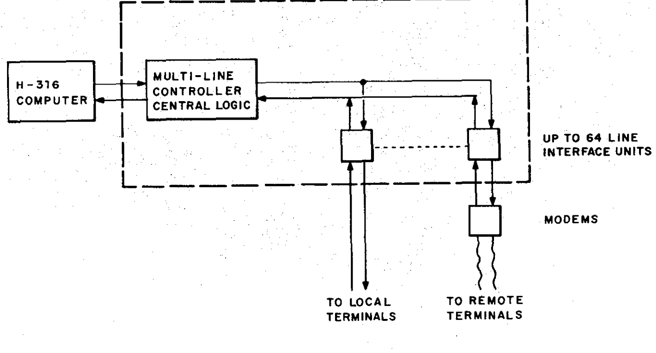


Figure 6—Block diagram of TIP hardware

cable, the ports may go to completely separate devices of entirely different properties. Thus, *input* port 16 may connect to a slow, asynchronous, 5-bit character keyboard while *output* port 16 connects to a high speed, synchronous display of some sort. In order to achieve this flexibility, the MLC stores information about each input and each output port and the program sets up this information for each half of each port in turn as it turns the ports "ON."

Several aspects of the MLC design are noteworthy. The central logic treats each of the 64 ports in succession, each port getting 800 ns of attention per cycle. The port then waits the remainder of the cycle (51.2 μs) for its next turn. For both input and output, two full characters are buffered in the central logic, the one currently being assembled or disassembled and one further character to allow for delays in memory accessing.

During input, characters from the various lines stream into a tumble table in memory on a first come, first served basis. Periodically a clock interrupt causes the program to switch tables and look for input.

Output characters are fed to all lines from a single output table. Ordering the characters in this table in such a way as to keep a set of lines of widely diverse speeds solidly occupied is a difficult task. To assist the program in this, a novel mechanism has been built into the MLC hardware whereby each line, as it uses up a character from the output table, enters a request consisting of its line number into a "request" table in memory. This table is periodically inspected by the program and the requests are used in building the next output table with the characters in proper line sequence.

The design of the terminal interface portion of the MLC is modular. Each Line Interface Unit (LIU) contains all the logic required for full duplex, bit serial communication and consists of a basic bi-directional data section and a control and status section. The data section contains transmit and receive portions each with clock and data lines. For asynchronous devices the clock line is ignored and timing is provided by the MLC itself. (For received asynchronous characters, timing is triggered by the leading edge of the start bit of each character.)

The control and status monitor functions are provided for modems as required by the RS-232C specification. Four outputs are available for control functions and six inputs are available to monitor status. The outputs are under program control and are available for non-standard functions if the data terminal is not a modem. For example, these lines could be used to operate a local line printer. RS-232C connectors are mounted directly on the LIU cards. To allow for variations in terminal and modem pin assignments, the signals are brought to connector pins via jumpers on the card.

The central MLC contains 256 ICs, many of which are MSI and some of which are LSI circuits, and it is thus about the same complexity as the basic H-316. In addition each LIU contains 31 ICs. A Terminal IMP including the MLC and with a typical interface configuration to high-speed circuits and Hosts is, order of magnitude, a $100,000 device.

THE TERMINAL USER'S VIEW

This section describes how a TIP user gains access to the network. The protocol described is of very recent origin and will undoubtedly change in response to usage which, as of this writing, is just commencing.

In general a user must have some foreknowledge of the resource which he expects to access via the network. The TIP program implements a set of commands[8] for connecting to and disconnecting from remote sites, but once a terminal is connected to a particular system, the TIP becomes transparent to the conversation unless specially signalled. This is equivalent to a timesharing system where the executive program is essentially out of the picture during periods while the user is dealing directly with his own set of programs.

Because of the large number of different terminal types used in the network, the concept of the Network Virtual Terminal was developed. This is an imaginary but well-defined type of terminal. The TIP translates typed data to virtual terminal code before shipping it into the network, and conversely translates the remote system's response back into the local terminal's code. Thus, each Host system must deal only with this single terminal type.

When the user at a terminal needs to talk directly to his TIP instead of to the remote Host, he issues a command which is distinguished by the fact that it always starts with the symbol @. One or more words, perhaps followed by a single parameter, then identify the type of command.

Normally a user will go through four more or less distinct stages in typing into the net. First, he will be concerned with hardware, power, dialing in, etc. Then he will establish a dialogue with the TIP to get a comfortable set of parameters for his usage. Next, he will instruct the TIP to make a connection to a remote Host, and finally, he will mostly ignore the TIP as he talks to the remote Host.

One of the more interesting features of the TIP is that it permits great flexibility in the types of terminals which may be attached to any port. This

TABLE II—Tip Commands

CLOSE	close all outstanding connections
HOST #	focus attention on this host 0<#<256
LOGIN	start the initial connection procedure to get Telnet connections
SEND BREAK	send a Telnet break character
SEND SYNC	send a Telnet sync character and an INTERRUPT SENDER message
ECHO ALL	local TIP-generated echo—TIP echoes everything
ECHO NONE	remote Host-generated echo—TIP will echo commands
ECHO HALFDUPLEX	terminal-generated echo—TIP echoes nothing
FLUSH	delete all characters in input buffer
TRANSMIT EVERY #	send off input buffer at least every #th character 0<#<256
TRANSMIT ON EVERY CHARACTER	like TRANSMIT EVERY 1
TRANSMIT ON LINEFEED	send input buffer every time a linefeed encountered
TRANSMIT ON MESSAGE-END	send input buffer every time an EOM encountered
TRANSMIT ON NO CHARACTER	do not transmit on linefeed or EOM
TRANSMIT NOW	send off input buffer now
DEVICE RATE #	# is a 13 bit code specifying hardware rate and character size settings
DEVICE CODE ASCII	establish code conversion
DEVICE CODE 2741	
DEVICE CODE EXECUPORT	
DEVICE CODE ODEC	
SEND TO HOST #	establish parameters for manual initiation of connections
RECEIVE FROM HOST #	
SEND TO SOCKET #	
RECEIVE FROM SOCKET #	
PROTOCOL TO TRANSMIT	initiate connection protocol manually
PROTOCOL TO RECEIVE	
PROTOCOL TO CLOSE TRANSMIT	
PROTOCOL TO COSE RECEIVE	
PROTOCOL BOTH	abbreviations for simultaneous transmit and receive protocols
PROTOCOL TO CLOSE BOTH	
# GIVE BACK	release control of captured device
# DIVERT OUTPUT	capture device # and divert this terminal's output to it
ABORT LOGIN	abort the outstanding initial connection procedure

flexibility presents a problem to the program which must determine what kind of terminal (speed, character size, etc.) is attached to a given port before a sensible exchange of information is possible. To solve this problem, each terminal is assigned a special identifying character which the user types repeatedly when starting to use a terminal. When information starts to appear from a previously idle port, the TIP enters a "hunt" mode in which it interprets the arriving characters trying various combinations of terminal characteristics. All except the proper combination will cause the character to be garbled, producing something other than one of the legal terminal identifying characters. When the TIP thus identifies the correct set (which generally requires the user to repeat the identifier less than half a dozen times), it types out ‘HELLO in the terminal's own language.

In the next stage, the user initializes certain conversation parameters relating to message size and when to echo. He then establishes connection to a remote site using two commands which identify the desired remote site and the fact that the user wishes to be connected to the logger at that site. These commands are:

@ HOST 15

@ LOGIN

The LOGIN command actually sets in motion an elaborate exchange of messages between the TIP and the remote Host which normally result in a connection being made to that remote system. This command will be answered by an appropriate comment to the user indicating either that a connection has been made, that the remote site is not up, that it is up but actively refusing to converse, or that it is up but not responding enough even to refuse the connection.

Once a connection is established, the user types directly to the logger. The TIP does not execute the actual login since this procedure varies from site to site.

Throughout the user-to-Host dialogue, commands remain available at any time. Prior to closing the connection, the user must log out as required by the system he is using. He then gives the command

@ CLOSE

which causes the TIP to close the connection, informing the user when the process is finished.

In addition to the above more or less standard procedures, there are a number of less usual commands which set such things as device rate, character size, code types, etc. Such commands are used by a terminal on one port in setting parameters for a non-interactive terminal, such as a printer or card reader, on some other port. Other special commands permit conversations directly between terminals on the same or different TIPs, allow for binary mode, etc. Table II is a list of the commands with a brief explanation. For details refer to Reference 8. Figure 7 gives an example of typical dialogue.

THE SOFTWARE

Because the terminals connected to a TIP communicate with Hosts at remote sites, the TIP, in addition to performing the IMP function, also acts as intermediary between the terminal and the distant Host. This means that network standards for format and protocol must be implemented in the TIP. One can thus think of the TIP software as containing both a very simpleminded mini-Host and a regular IMP program.

Figure 8 gives a simplified diagrammatic view of the program. The lower block marked "IMP" represents the usual IMP program. The two lines into and out of that block are logically equivalent to input and output from a Host. The code conversion blocks are in fact surprisingly complex and include all of the material for dealing with diverse (and perverse) types of terminals.

As the user types on the keyboard, characters go, via input code conversion, to the input block. Information for remote sites is formed into regular network messages and passes through the OR switch to the IMP program for transmittal. Command characters are fed off to the side to the command block where commands are decoded. The commands are then "performed" in that they either set some appropriate parameter or a flag which calls for later action. An example of this is the LOGIN command. Such a command in fact triggers a complex network protocol procedure, the various steps of which are performed by the PROTOCOL block working in conjunction with the remote Host through the IMPs. As part of this process an appropriate special message will be sent to the terminal via the Special Messages block indicating the status (success, failure, etc.) of the procedure.

```
1               Character typed just after terminal turned on.  Used to
                identify the terminal type for the Terminal IMP program.

HELLO           TIP indicates that terminal is ready for use.

@ HOST 1        User tells Terminal IMP which Host to connect to.

                (Terminal IMP echoes extra carriage return line feed to
                indicate it has accepted a command.)
@ LOGIN         User tells Terminal IMP to form a connection.
                (TIP echoes extra carriage return line feed.)

T R OPEN        Terminal IMP has formed a connection.

LOG ON*         This is UCLA Σ7's greeting message.          ⎫
                                                             ⎪
XXXX            User identification.                         ⎪
                         :                                   ⎪
!                        :                                   ⎪
                         :                                   ⎪
S* .ABACUS:C             :                                   ⎪
                         :                                   ⎬ Dialogue between
.                        :                                   ⎪ terminal user
.                        :                                   ⎪ and UCLA Σ7.
.                        :                                   ⎪
X                        :                                   ⎪
                         :                                   ⎪
STOP                     :                                   ⎪
                         :                                   ⎪
NORMAL EXIT              :                                   ⎪
                         :                                   ⎪
!                        :                                   ⎪
                                                             ⎪
X               User exits from UCLA Σ7.                     ⎪
                                                             ⎪
LOG ON*         Σ7 closing message indicates user has logged off.  ⎭

@ CLOSE         User detaches the Terminal IMP from computer being used
                (i.e. closes connection).
                (TIP echoes extra carriage return line feed.)
T R CLOSED      Terminal IMP has closed the connection.
```

Figure 7—Typical terminal user dialogue

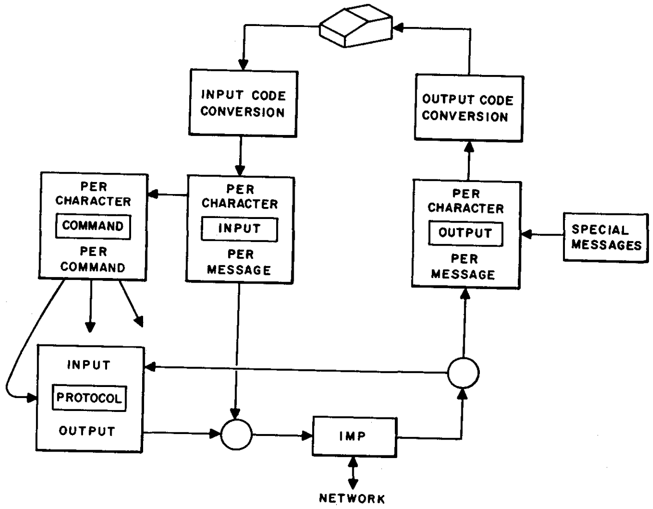


Figure 8—Block diagram of TIP program

Once connection to a remote Host is established, regular messages flow directly through the Input block and on through the IMP program. Returning responses come in through the IMP, into the OUTPUT program where they are fed through the OUTPUT Code Conversion block to the terminal itself.

Storage for the TIP program, including the standard IMP program, is just over 20K. This is roughly as follows:

Standard IMP Program with buffers & tables	12,000
Special TIP code	2,650
Tables of Parameters	1,800
Buffer Storage for messages to and from terminals	1,880
Miscellaneous—I/O buffers, constants, etc.	1,880

PERFORMANCE

The program can handle approximately 100 kilobits of one-way terminal traffic provided the message size is sufficiently large that per-message processing is amortized over many characters. Overhead per message is such that if individual characters are sent one at a time there is a loss of somewhere between a factor of ten and twenty in bandwidth. A different way to look at program performance is to observe that the per-character processing time is about 75 μs.

These figures ignore the fact that the machine must devote some of its bandwidth to acting as an IMP, both for terminal traffic and for regular network traffic. About 5% of the machine is lost to acting as an IMP, even in the absence of traffic. If there is network traffic, more of the machine bandwidth is used up. Five hundred kilobits of two-way phone line and Host traffic saturates the machine without any terminal traffic*.

In addition to bandwidth which goes into the IMP part of the job, another 10 percent of the (total) machine is taken up simply in fielding clock interrupts from the Multi-Line Controller. This again is bandwidth used in idling even with no actual terminal traffic.

The following formula summarizes, approximately, the bandwidth capabilities:

$$P + H + 11T \leq 850$$

where:

P = total phone line traffic (in kilobits/sec) wherein, for example, a full duplex 50 Kb phone line counts as 100;

H = total Host traffic (in kilobits/sec) wherein the usual full duplex Host interface, with its usual 10 μs/bit setting, counts as 200; and

T = total terminal traffic (in kilobits/sec) wherein an ASCII terminal such as a (110-baud) full-duplex Teletype (ASR-33) counts as twice its baud rate (i.e., 0.220 Kb).

This means that it takes eleven times as much program time to service every terminal character as it does to service a character's worth of phone line or Host message.

A further factor that influences terminal traffic handling capability has to do with the terminals themselves. Certain types of terminals require more attention from the program than others, independent of their speed but based rather on their complexity. In particular, for example, while an IBM 2741 nominally runs at 134.5 bits per second, the complexity is such that it uses nearly three times the program bandwidth that would be used in servicing a half-duplex ASCII terminal of equivalent speed. Allowances for such variations must be made in computing the machine's ability to service a particular configuration. It must be borne in mind that all of these performance figures are approximations and that the actual rules are extremely complex, and will change as the program matures.

DISCUSSION

As the ARPA Network grows, a number of areas of development seem likely to require attention. Certain of these are already pressing problems whereas others will begin to appear primarily as the network continues to grow and mature.

Perhaps the most difficult aspect of a network such as this is the problem of reliability. A great deal of thought and effort has gone into trying to make the system reliable and network topology has been designed with the need for redundancy in mind. Nonetheless the problem of keeping a large number of computer systems going at widely separated sites is non-trivial. An IMP's mean-time-between-failure (MTBF) has been on the order of one month; considerably lower than expected. The problems arise from a variety of sources and the system is sufficiently complex that frequently the cause has been masked by the time the failure has been noted. Often the same failure will recur several times until the problem is identified and eliminated and this fact tends to decrease the apparent MTBF. In general a preponderance of the troubles stem from hardware failures, and we are currently modifying several noisy and/or marginal portions of the I/O and interface hardware in hopes of obtaining significant improvement.

Our strategy has generally been to spend time identifying the source of trouble, keeping the machine off the air if necessary, so that eventually the MTBF will increase. This has often meant, however, that a "down" was much longer than it would have been, had the foremost goal been to get the machine running again immediately. With this strategy the average down time has been about 9 hours, giving rise to an average per-machine down rate of about 2 percent. We hope to improve this situation in the near future by providing improved facilities for obtaining "program dumps" when a failure occurs. This will make it possible to bring machines back on the air almost immediately in many cases, without jeopardizing valuable debugging data.

* The number 500 kilobits is for full size (8000 bit) messages. Shorter messages use up more capability per bit and thus reduce the overall bandwidth capability.

A word about our experience with the phone lines appears appropriate here as well. We have apparently been an unusual, if not unique, customer for the common carriers in our degree of attention to line failures. Our ability to detect and report even brief outages has led through measured skepticism to eventual acceptance by the carriers. In general, tests have indicated that the phone line error rates are about as predicted, i.e., approximately one in 10^5. These occasional errors or error bursts do not appear to affect network performance adversely. The IMP-to-IMP error checking procedure has not, to our knowledge, admitted a single erroneous message. We have, however, had some difficulty with lines which were simply out of commission from time to time for extended periods (hours or even days). The reliability of the phone lines is roughly equivalent to the reliability of the IMPs, based on the number and duration of outages. Down times have decreased as the carriers have come to accept our complaints as generally legitimate. *Overall the performance of the telephone equipment does not appear to constitute a problem in network growth.*

From a strictly technical viewpoint we view the incorporation of higher bandwidth facilities as a natural and key part of network growth. While the present facilities are not saturated by the present loads, we view this situation as a temporary one and something of a period of grace. As the network continues to grow over the next few years traffic can be expected to increase in a very non-linear fashion. Host protocol procedures (which have presented a sizable stumbling block to usage) are settling down so that software commitments can be made, and the advent of the Terminal IMP will bring an influx of new users who do not have the alternative of local resources. As a result we expect that traffic will begin to saturate some parts of the network. Terminal IMPs may well be called upon to service a larger number of terminals of higher bandwidth than can be handled by the present version.

In anticipation of these requirements we are presently considering the design of a significantly faster and more modular version of the IMP. Its higher speed will permit it to take advantage of high speed (i.e., megabit and above) communication lines which the common carriers are currently developing. More generally it will be able to service high bandwidth requirements whenever and however they occur within the net. We are currently studying a modular design which will permit connection of a greater number of interfaces than the present IMP can handle. While this work is only in the preliminary design stage, we feel that the interest and enthusiasm which have greeted the initial network suggest that it is not too early to consider ways to cope with growing traffic requirements.

Another area of network development which has already received a great deal of attention and will require much more in the future is that of technical information interchange between the sites. To the user, the resources which are becoming available are staggering if not overwhelming. It is very difficult for an individual to discover what, if any, of the mass of programs that will be available through the network may be of interest or of use to him. To aid in this process a number of mechanisms are being implemented and we are cooperating closely with these efforts. In particular, at the Stanford Research Institute a Network Information Center (NIC) acts essentially as a library for documentation concerning the network. Here are maintained a number of pointer documents, specifically, (1) a Directory of Participants which lists critical personnel, phone numbers, etc., for each participating site, (2) a catalogue of the NIC Collection which lists the available documents indexed in a variety of ways, (3) a Resource Notebook which is a compendium that attempts to familiarize the reader with available program resources at each of the participating sites.

Finally, there is the broad and exciting issue of how to cope with success. As the ARPA Network grows, and as diverse resources and users join the net, it is clear that a technology transfer must occur; the network probably must shift from a government-supported research and development activity to an ongoing national *service* of some kind. However, the computer-communications environment in the U.S. is rather complex, and is populated with many legal, economic, and political constraints. Within this environment, it is not easy to perform the technology transfer, and many groups, including ARPA and BBN, have been considering possible alternative plans. We are optimistic that a way will be found to provide a suitable legal and political base for expansion of the present ARPA Network into a widely useful public communication system.

ACKNOWLEDGMENTS

In addition to the authors, a great many people have contributed to the work reported here. Dr. L. G. Roberts of the Advanced Research Projects Agency has continued to lend crucial support and encouragement to the project. Messrs. Blue, Dolan, and Crocker of the ARPA office have been understanding and helpful. Suggestions and helpful criticism have come from many present and prospective participants in the network community.

At BBN, W. B. Barker is responsible for much of the cleverness which appears in the MLC hardware; R. E. Kahn has contributed in many areas and has been deeply involved in studying traffic flow control; A. McKenzie has compiled the Network Resource Notebook and been concerned with Host relations and many Host protocol issues; M. Thrope has managed the Network Control Center and has kept the network on the air; J. Levin helped in implementation of the TIP software; B. P. Cosell has patiently labored with the IMP software, with help from J. McQuillan and M. Kraley; and R. Alter has offered much helpful advice and criticism.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

2 F E HEART R E KAHN S M ORNSTEIN W R CROWTHER D C WALDEN
The interface message processor for the ARPA computer network
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

3 L KLEINROCK
Analytic and simulation methods in computer network design
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

4 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

5 C S CARR S D CROCKER V G CERF
Host-host communication protocol in the ARPA network
AFIPS Conference Proceedings Spring Joint Computer Conference 1970

6 S CROCKER et al
Function-oriented protocols for the ARPA computer network
AFIPS Conference Proceedings Spring Joint Computer Conference 1972

7 R E KAHN W R CROWTHER
Flow control in a resource sharing computer network
Proc Second Symposium on Problems in the Optimization of Data Communications Systems October 1971

8 *User's guide to the terminal IMP*
Bolt Beranek and Newman Inc Report No 2183

9 *The BBN terminal interface message processor*
BBN Report 2184

10 L G ROBERTS B D WESSLER
The ARPA network
Computer Communication Networks Chapter 6 In preparation

11 L G ROBERTS
A forward look
Journal of Armed Forces Communications and Electronics Assoc Vol XXV No 12 August 1971 pp 77-81

12 *The MIT Lincoln Lab terminal support processor*
Graphics Semi-Annual Tech Summary Reports ESD-TR-70-151 p 355 May November 1970

13 *Progress report #1*
University of Michigan MERIT Computer Network Report #0571PR4 May 1971

14 A B COCANOWER W FISCHER W S GERSTENBERGER B S READ
The communications computer operating system—The initial design
University of Michigan MERIT Computer Network Manual #1070-TN-3 October 1970

15 R E KAHN
Terminal access to the ARPA computer network
Courant Computer Symposium 3 Computer Networks Courant Institute New York November 1970—Proceedings to be published by Prentice Hall Englewood Cliffs New Jersey In preparation

Computer communication network design—Experience with theory and practice*

by HOWARD FRANK
Network Analysis Corporation
Glen Cove, New York

ROBERT E. KAHN
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

and

LEONARD KLEINROCK
University of California
Los Angeles, California

INTRODUCTION

The ARPA Network (ARPANET) project brought together many individuals with diverse backgrounds, philosophies, and technical approaches from the fields of computer science, communication theory, operations research and others. The project was aimed at providing an efficient and reliable computer communications system (using message switching techniques) in which computer resources such as programs, data, storage, special purpose hardware etc., could be shared among computers and among many users.[38] The variety of design methods, ranging from theoretical modeling to hardware development, were primarily employed independently, although cooperative efforts among designers occurred on occasion. As of November, 1971, the network has been an operational facility for many months, with about 20 participating sites, a network information center accessible via the net, and well over a hundred researchers, system programmers, computer center directors and other technical and administrative personnel involved in its operation.

In this paper, we review and evaluate the methods used in the ARPANET design from the vantage of over two years' experience in the development of the network. In writing this paper, the authors have each made equal contributions during a series of intensive discussions and debates. Rather than present merely a summary of the procedures that were used in the network design, we have attempted to evaluate each other's methods to determine their advantages and drawbacks. Our approaches and philosophies have often differed radically and, as a result, this has not been an easy or undisturbing process. On the other hand, we have found our collaboration to be extremely rewarding and, notably, we have arrived at many similar conclusions about the network's behavior that seem to be generally applicable to message switched networks.

The essence of a network is its design philosophy, its performance characteristics, and its cost of implementation and operation. Unfortunately, there is no generally accepted definition of an "optimal" network or even of a "good" network. For example, a network designed to transmit large amounts of data only during late evening hours might call for structural and performance characteristics far different from one servicing large numbers of users who are rapidly exchanging short messages during business hours. We expect this topic, and others such as the merits of message switching vs. circuit switching or distributed vs. centralized control to be a subject of discussion for many years.[1,14,24,32,34,37]

A cost analysis performed in 1967-1968 for the ARPA Network indicated that the use of message switching would lead to more economical communications and better overall availability and utilization of resources than other methods.[36,38] In addition to its impact on the availability of computer resources, this decision has generated widespread interest in store-and-forward communications. In many instances, the use of store-and-forward communication techniques can result in

* This work was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC 15-70-C-0120 at the Network Analysis Corporation, Contract Nos. DAHC 15-69-C-0179 and DAHC-71-C-0088 at Bolt Beranek and Newman Inc., and Contract No. DAHC 15-69-C-0285 at the University of California at Los Angeles.

greater flexibility, higher reliability, significant technical advantage, and substantial economic savings over the use of conventional common carrier offerings. An obvious trend toward increased computer and communication interaction has begun. In addition to the ARPANET, research in several laboratories is under way, small experimental networks are being built, and a few examples of other government and commercial networks are already apparent.[6,7,31,40,41,47,48,52]

In the ARPANET, each time-sharing or batch processing computer, called a Host, is connected to a small computer called an Interface Message Processor (IMP). The IMPs, which are interconnected by leased 50 kilobit/second circuits,' handle all network communication for their Hosts. To send a message to another Host, a Host precedes the text of its message with an address and simply delivers it to its IMP. The IMPs then determine the route, provide error control, and notify the sender of its receipt. The collection of Hosts, IMPs, and circuits forms the message switched resource sharing network. A good description of the ARPANET, and some early results on protocol development and modeling are given in References 3, 12, 15, 23 and 38. Some experimental utilization of the ARPANET is described in Reference 42. A more recent evaluation of such networks and a forward look is given in References 35 and 39.

The development of the Network involved four principal activities:

(1) The design of the IMPs to act as nodal store-and-forward switches,
(2) The topological design to specify the capacity and location of each communication circuit within the network,
(3) The design of higher level protocols for the use of the network by time-sharing, batch processing and other data processing systems, and
(4) System modeling and measurement of network performance.

Each of the first three activities were essentially performed independently of each other, whereas the modeling effort partly affected the IMP design effort, and closely interacted with the topological design project.

The IMPs were designed by Bolt Beranek and Newman Inc. (BBN) and were built to operate independent of the exact network connectivity; the topological structure was specified by Network Analysis Corporation (NAC) using models of network performance developed by NAC and by the University of California at Los Angeles (UCLA). The major efforts in the area of system modeling were performed at UCLA using theoretical and simulation techniques. Network performance measurements have been conducted during the development of the network by BBN and by the Network Measurement Center at UCLA. To facilitate effective use of the net, higher level (user) protocols are under development by a group of representatives of universities and research centers. This group, known as the Network Working Group, has already specified a Host to Host protocol and a Telnet protocol, and is in the process of completing other function oriented protocols.[4,29] We make no attempt to elaborate on the Host to Host protocol design problems in this paper.

THE NETWORK DESIGN PROBLEM

A variety of performance requirements and system constraints were considered in the design of the net. Unfortunately, many of the key design objectives had to be specified long before the actual user requirements could be known. Once the decision to employ message switching was made, and fifty kilobit/second circuits were chosen, the critical design variables were the network operating procedure and the network topology; the desired values of throughput, delay, reliability and cost were system performance and constraint variables. Other constraints affected the structure of the network, but not its overall properties, such as those arising from decisions about the length of time a message could remain within the network, the location of IMPs relative to location of Hosts, and the number of Hosts to be handled by a single IMP.

In this section, we identify the central issues related to IMP design, topological design, and network modeling. In the remainder of the paper, we describe the major design techniques which have evolved.

IMP properties

The key issue in the design of the IMPs was the definition of a relationship between the IMP subnet and the Hosts to partition responsibilities so that reliable and efficient operation would be achieved. The decision was made to build an autonomous subnet, independent (as much as possible) of the operation of any Host. The subnet was designed to function as a "communications system"; issues concerning the use of the subnet by the Hosts (such as protocol development) were initially left to the Hosts. For reliability, the IMPs were designed to be robust against all line failures and the vast majority of IMP and Host failures. This decision required routing strategies that dynamically adapt to changes in the states of IMPs and circuits, and an elaborate flow control strategy to protect the subnet against Host malfunction and congestion due to IMP buffer limitations. In addition, a statistics and status reporting mechanism was needed to monitor the behavior of the network.

The number of circuits that an IMP must handle is a design constraint directly affecting both the structure of the IMP and the topological design. The speed of the IMP and the required storage for program and buffers depend directly upon the total required processing capacity, which must be high enough to switch traffic from one line to another when all are fully occupied. Of great importance is the property that all IMPs operate with identical programs. This technique greatly simplifies the problem of network planning and maintenance and makes network modifications easy to perform.

The detailed physical structure of the IMP is not discussed in this paper.[2,15] However, the operating procedure, which guides packets through the net, is very much of interest here. The flow control, routing, and error control techniques are integral parts of the operating procedure and can be studied apart from the hardware by which they are implemented. Most hardware modifications require changes to many IMPs already installed in the field, while a change in the operating procedure can often be made more conveniently by a change to the single operating program common to all IMPs, which can then be propagated from a single location via the net.

Topological properties

The topological design resulted in the specification of the location and capacity of all circuits in the network. Projected Host—Host traffic estimates were known at the start to be either unreliable or wrong. Therefore, the network was designed under the assumption of equal traffic between all pairs of nodes. (Additional superimposed traffic was sometimes included for those nodes with expectation of higher traffic requirements.) The topological structure was determined with the aid of specially developed heuristic programs to achieve a low cost, reliable network with a high throughput and a general insensitivity to the exact traffic distribution. Currently, only 50 kilobit/second circuits are being used in the ARPANET. This speed line was chosen to allow rapid transmission of short messages for interactive processing (e.g., less than 0.2 seconds average packet delay) as well as to achieve high throughput (e.g., at least 50 kilobits/second) for transmission of long messages. For reliability, the network was constrained to have at least two independent paths between each pair of IMPs.

The topological design problem requires consideration of the following two questions:

(1) Starting with a given state of the network topology, what circuit modifications are required to add or delete a set of IMPs?

(2) Starting with a given state of network topology, when and where should circuits be added or deleted to account for long term changes in network traffic?

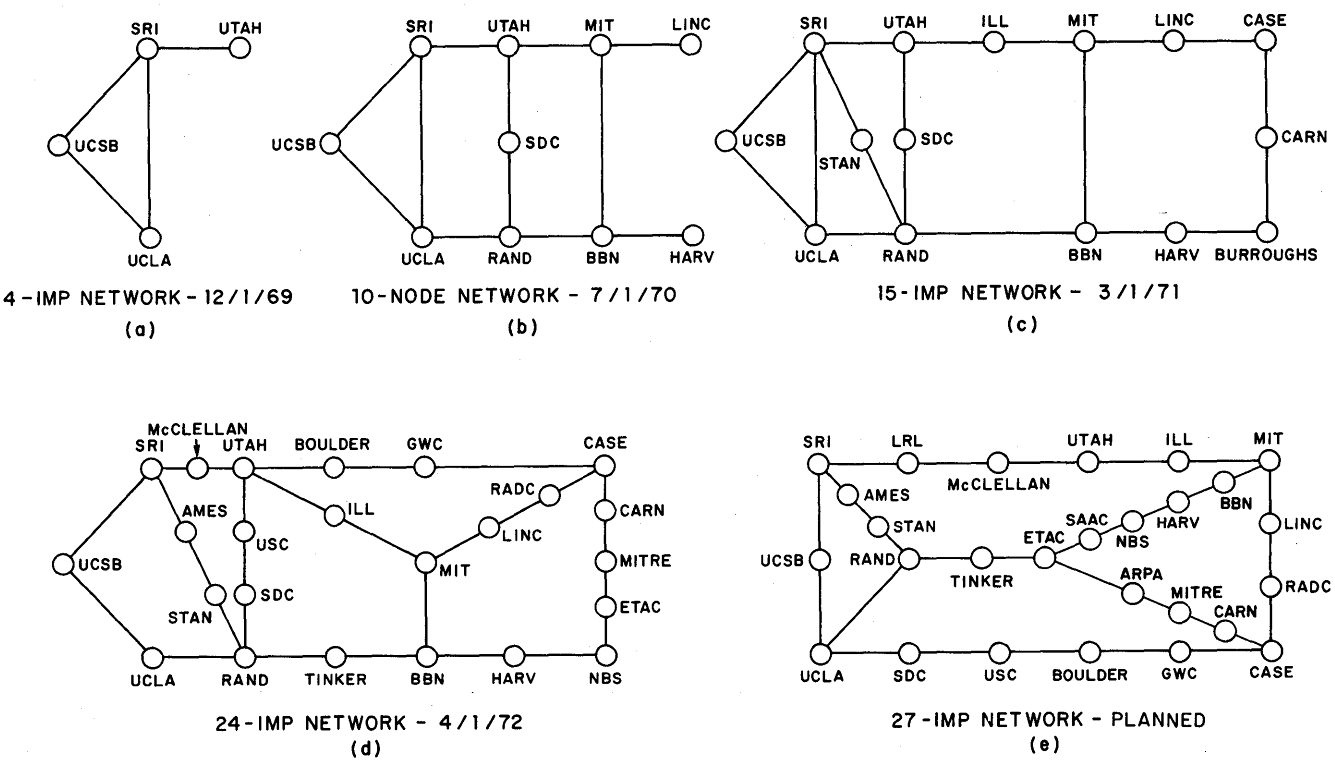
If the locations of all network nodes are known in advance, it is clearly most efficient to design the topological structure as a single global effort. However, in the ARPANET, as in most actual networks, the initial designation of node locations is modified on numerous occasions. On each such occasion, the topology can be completely reoptimized to determine a new set of circuit locations.

In practice, there is a long lead time between the ordering and the delivery of a circuit, and major topological modifications cannot be made without substantial difficulty. It is therefore prudent to add or delete nodes with as little disturbance as possible to the basic network structure consistent with overall economical operation. Figure 1 shows the evolution of the ARPANET from the basic four IMP design in 1969 to the presently planned 27 IMP version. Inspection of the 24 and 27 IMP network designs reveals a few substantial changes in topology that take advantage of the new nodes being added. Surprisingly enough, a complete "reoptimization" of the 27 IMP topology yields a network only slightly less expensive (about 1 percent) than the present network design.[28]

Network models

The development of an accurate mathematical model for the evaluation of time delay in computer networks is among the more difficult of the topics discussed in this paper. On the one hand, the model must properly reflect the relevant features of the network structure and operation, including practical constraints. On the other hand, the model must result in a mathematical formulation which is tractable and from which meaningful results can be extracted. However, the two requirements are often incompatible and we search for an acceptable compromise between these two extremes.

The major modeling effort thus far has been the study of the behavior of networks of queues.[21] This emphasis is logical since in message switched systems, messages experience queueing delays as they pass from node to node and thus a significant performance measure is the



4-IMP NETWORK - 12/1/69
(a)

10-NODE NETWORK - 7/1/70
(b)

15-IMP NETWORK - 3/1/71
(c)

24-IMP NETWORK - 4/1/72
(d)

27-IMP NETWORK - PLANNED
(e)

Figure 1—The evolution of the ARPANET

speed at which messages can be delivered. The queueing models were developed at a time when there were no operational networks available for experimentation and model validation, and simulation was the only tool capable of testing their validity. The models, which at all times were recognized to be idealized statements about the real network, were nonetheless crucial to the ARPANET topological design effort since they afforded the only known way to quantitatively predict the properties of different routing schemes and topological structures. The models have been subsequently demonstrated to be very accurate predictors of network throughput and indispensable in providing analytical insight into the network's behavior.

The key to the successful development of tractable models has been to factor the problem into a set of simpler queueing problems. There are also heuristic design procedures that one can use in this case. These procedures seem to work quite well and are described later in the paper. However, if one specializes the problem and removes some of the real constraints, theory and analysis become useful to provide understanding, intuition and design guidelines for the original constrained problem. This approach uncovers global properties of network behavior, which provide keys to good heuristic design procedures and ideal performance bounds.

DESIGN TECHNIQUES

In this section we describe the approaches taken to the design problems introduced in the previous section. We first summarize the important properties of the ARPANET design:

(1) A communications cost of less than 30 cents per thousand packets (approximately a megabit).
(2) Average packet delays under 0.2 seconds through the net.
(3) Capacity for expansion to 64 IMPs without major hardware or software redesign.
(4) Average total throughput capability of 10-15 kilobits/second for all Hosts at an IMP.
(5) Peak throughput capability of 85 kilobits/second per pair of IMPs in an otherwise unloaded network.
(6) Transparent communications with maximum message size of approximately 8000 bits and error rates of one bit in 10^{12} or less.

(7) Approximately 98 percent availability of any IMP and close to 100 percent availability of all operating IMPs from any operable IMP.

The relationships between the various design efforts are illustrated by these properties. The topological design provides for both a desired average throughput and for two or more paths to be fully used for communication between any pair of Hosts. The operating procedure should allow any pair of Hosts to achieve those objectives. The availability of IMPs to communicate reflects both the fact that IMPs are down about 2 percent of the time and that the topology is selected so that circuit failures contribute little additional to the total system downtime.

IMP design

The IMP design consists of two closely coupled but nonetheless separable pieces—the physical hardware specification (based on timing and reliability considerations and the operating procedure) and the design and implementation of the operating procedure using the specified IMP hardware. The IMP originally developed for the ARPANET contains a 16-bit one microsecond computer that can handle a total of about ¾ megabits/second of "useful" information on a total of approximately one megabit/second of circuit capacity (e.g., twenty 50 kilobit/second circuits). Hardware is likely to change as a function of the required IMP capacity but an operating procedure that operates well at one IMP capacity is likely to be transferable to machines that provide different capacity. However, as a network grows in size and utilization, a more comprehensive operating procedure that takes account of known structural properties, such as a hierarchical topology, is appropriate.

Four primary areas of IMP design, namely message handling and buffering, error control, flow control, and routing are discussed in this section. The IMP provides buffering to handle messages for its Host and packets for other IMPs. Error control is required to provide reliable communication of Host messages in the presence of noisy communication circuits. The design of the operating procedure should allow high throughput in the net under heavy traffic loads. Two potential obstacles to achieving this objective are: (1) The net can become congested and cause the throughput to *decrease* with increasing load, and (2) The routing procedure may be unable to always adapt sufficiently fast to the rapid movement of packets to insure efficient routing. A flow control and routing procedure is needed that can efficiently meet this requirement.

Message handling and buffering

In the ARPANET, the maximum message size was constrained to be approximately 8000 bits. A pair of Hosts will typically communicate over the net via a sequence of transmitted messages. To obtain delays of a few tenths of a second for such messages and to lower the required IMP buffer storage, the IMP program partitions each message into one or more packets each containing at most approximately 1000 bits. Each packet of a message is transmitted independently to the destination where the message is reassembled by the IMP before shipment to that destination Host. Alternately, the Hosts could assume the responsibility for reassembling messages. For an asynchronous IMP-Host channel, this marginally simplifies the IMP's task. However, if *every* IMP-Host channel were synchronous, and the Host provided the reassembly, the IMP task can be further simplified. In this latter case, "IMP-like" software would have to be provided in each Host.

The method of handling buffers should be simple to allow for fast processing and a small amount of program. The number of buffers should be sufficient to store enough packets for the circuits to be used to capacity; the size of the buffers may be intuitively selected with the aid of simple analytical techniques. For example, fixed buffer sizes are useful in the IMP for simplicity of design and speed of operation, but inefficient utilization can arise because of variable length packets. If each buffer contains A words of overhead and provides space for M words of text, and if message sizes are uniformly distributed between 1 and L, it can be shown[45] that the choice of M that minimizes the expected storage is approximately $\sqrt{AL}$. In practice, M is chosen to be somewhat smaller on the assumption that most traffic will be short and that the amount of overhead can be as much as, say, 25 percent of buffer storage.

Error control

The IMPs must assume the responsibility for providing error control. There are four possibilities to consider:

(1) Messages are delivered to their destination out of order.

(2) Duplicate messages are delivered to the destination.

(3) Messages are delivered with errors.

(4) Messages are not delivered.

The task of proper sequencing of messages for delivery to the destination Host actually falls in the province of both error control and flow control. If at most one message at a time is allowed in the net between a pair of Hosts, proper sequencing occurs naturally. A duplicate packet will arrive at the destination IMP after an acknowledgment has been missed, thus causing a successfully received packet to be retransmitted. The IMPs can handle the first two conditions by assigning a sequence number to each packet as it enters the network and processing the sequence number at the destination IMP. A Host that performs reassembly can also assign and process sequence numbers and check for duplicate packets. For many applications, the order of delivery to the destination is immaterial. For priority messages, however, it is typically the case that fast delivery requires a perturbation to the sequence.

Errors are primarily caused by noise on the communication circuits and are handled most simply by error detection and retransmission between each pair of IMPs along the transmission path. This technique requires extra storage in the IMP if either circuit speeds or circuit lengths substantially increase. Failures in detecting errors can be made to occur on the order of years to centuries apart with little extra overhead (20-30 parity bits per packet with the 50 kilobit/second circuits in the ARPANET). Standard cyclic error detection codes have been usefully applied here.

A reliable system design insures that each transmitted message is accurately delivered to its intended destination. The occasional time when an IMP fails and destroys a useful in-transit message is likely to occur far less often than a similar failure in the Hosts and has proven to be unimportant in practice, as are errors due to IMP memory failures. A simple end to end retransmission strategy will protect against these situations, if the practical need should arise. However, the IMPs are designed so that they can be removed from the network without destroying their internally stored packets.

Flow control

A network in which packets may freely enter and leave can become congested or logically deadlocked and cause the movement of traffic to halt.[5,17] Flow control techniques are required to prevent these conditions from occurring. The provision of extra buffer storage will mitigate against congestion and deadlocks, but cannot in general prevent them.

The sustained failure of a destination Host to accept packets from its IMP at the rate of arrival will cause the net to fill up and become congested. Two kinds of logical deadlocks, known as reassembly lockup and store-and-forward lockup may also occur. In reassembly lockup, the remaining packets of partially reassembled messages are blocked from reaching the destination IMP (thus preventing the message from being completed and the reassembly space freed) by other packets in the net that are waiting for reassembly space at that destination to become free. In a store-and-forward lockup, the destination has room to accept arriving packets, but the packets interfere with each other by tying up buffers in transit in such a way that none of the packets are able to reach the destination.[17] These phenomena have only been made to occur during very carefully arranged testing of the ARPANET and by simulation.[49]

In the original ARPANET design, the use of software links and RFNMS protected against congestion by a single link or a small set of links. However, the combined traffic on a large number of links could still produce congestion. Although this strategy did not protect against lockup, the method has provided ample protection for the levels of traffic encountered by the net to date.

A particularly simple flow control algorithm that augments the original IMP design to prevent congestion and lockup is also described in Reference 17. This scheme includes a mechanism whereby packets may be discarded from the net at the destination IMP when congestion is about to occur, with a copy of each discarded packet to be retransmitted a short time later by the originating Host's IMP. Rather than experience excessive delays within the net as traffic levels are increased, the traffic is queued outside the net so that the transit time delays internal to the net continue to remain small. This strategy prevents the insertion of more traffic into the net than it can handle.

It is important to note the dual requirement for small delays for interactive traffic and high bandwidth for the fast transfer of files. To allow high bandwidth between a pair of Hosts, the net must be able to accept a steady flow of packets from one Host and at the same time be able to rapidly quench the flow at the entrance to the source IMP in the event of imminent congestion at the destination. This usually requires that a separate provision be made in the algorithm to protect short interactive messages from experiencing unnecessarily high delays.

Routing

Network routing strategies for distributed networks require routing decisions to be made with only information available to an IMP and the IMP must execute those decisions to effect the routing.[14,15] A simple example of such a strategy is to have each IMP handling a packet independently route it along its current estimate of the shortest path to the destination.

For many applications, it suffices to deal with an idealized routing strategy which may not simulate the IMP routing functions in detail or which uses information not available to the IMP. The general properties of both strategies are usually similar, differing mainly in certain implementation details such as the availability of buffers or the constraint of counters and the need for the routing to quickly adapt to changes in IMP and circuit status.

The IMPs perform the routing computations using information received from other IMPs and local information such as the alive/dead state of its circuits. In the normal case of time varying loads, local information alone, such as the length of internal queues, is insufficient to provide an efficient routing strategy without assistance from the neighboring IMPs. It is possible to obtain sufficient information from the neighbors to provide efficient routing, with a small amount of computation needed per IMP and without each IMP requiring a topological map of the network. In certain applications where traffic patterns exhibit regularity, the use of a central controller might be preferable. However, for most applications which involve dynamically varying traffic flow, it appears that a central controller cannot be used more effectively than the IMPs to update routing tables if such a controller is constrained to derive its information via the net. It is also a less reliable approach to routing than to distribute the routing decisions among the IMPs.

The routing information cannot be propagated about the net in sufficient time to accurately characterize the instantaneous traffic flow. An efficient algorithm, therefore, should not focus on the movement of individual packets, but rather use topological or statistical information in the selection of routes. For example, by using an averaging procedure, the flow of traffic can be made to build up smoothly. This allows the routing algorithm ample time to adjust its tables in each IMP in advance of the build-up of traffic.

The scheme originally used in the ARPA network had each IMP select one output line per destination onto which to route packets. The line was chosen to be the one with minimum estimated time delay to the destination. The selection was updated every half second using minimum time estimates from the neighboring IMPs and internal estimates of the delay to each of the neighbors. Even though the routing algorithm only selects one line at a time per destination, two output lines will be used if a queue of packets waiting transmission on one line builds up before the routing update occurs and another line is chosen. Modifications to the scheme which allow several lines per destination to be used in an update interval (during which the routing is not changed) are possible using two or more time delay estimates to select the paths.

In practice, this approach has worked quite effectively with the moderate levels of traffic experienced in the net. For heavy traffic flow, this strategy may be inefficient, since the routing information is based on the length of queues, which we have seen can change much faster than the information about the change can be distributed. Fortunately, this information is still usable, although it can be substantially out of date and will not, in general, be helpful in making efficient routing decisions in the heavy traffic case.

A more intricate scheme, recently developed by BBN, allows multiple paths to be efficiently used even during heavy traffic.[18] Preliminary simulation studies indicate that it can be tailored to provide efficient routing in a network with a variety of heavy traffic conditions. This method separates the problem of defining routes onto which packets may be routed from the problem of selecting a route when a particular packet must be routed. By this technique, it is possible to send packets down a path with the fewest IMPs and excess capacity, or when that path is filled, the one with the next fewest IMPs and excess capacity, etc.

A similar approach to routing was independently derived by NAC using an idealized method that did not require the IMPs to participate in the routing decisions. Another approach using a flow deviation technique has recently been under study at UCLA.[13] The intricacies of the exact approach lead to a metering procedure that allows the overall network flow to be changed slowly for stability and to perturb existing flow patterns to obtain an increased flow. These approaches all possess, in common, essential ingredients of a desirable routing strategy.

Topological considerations

An efficient topological design provides a high throughput for a given cost. Although many measures of throughput are possible, a convenient one is the average amount of traffic that a single IMP can send into the network when all other IMPs are transmitting according to a specified traffic pattern. Often, it is assumed that all other IMPs are behaving identically and each IMP is sending equal amounts of traffic to each other IMP. The constraints on the topological design are the available common carrier circuits, the target cost or throughput, the desired reliability, and

TABLE I—23 Node 28 Link ARPA

Number of Circuits Failed	Number of Combinations to be Examined		Number of Cutsets
28	1		1
27	28		28
26	378		378
25	3276		3276
24	20475		20475
23	98280		98280
22	376740		376740
21	1184040		1184040
20	3108105		3108105
19	6906900		6906900
18	13123110	same	13123110
17	21474180		21474180
16	30421755		30421755
15	37442160		37442160
14	40116600		40116600
13	37442160		37442160
12	30421755		30421755
11	21474180		21474180
10	13123110		13123110
9	6906900		6906900
8	3108108		3108108
7	1184040		1184040
6	376740		349618
5	98280		≈70547
4	20475		≈9852
3	3276		827
2	378		30
1	28		0

the cost of computation required to perform the topological design.

Since, there was no clear specification of the amount of traffic that the network would have to accommodate initially, it was first constructed with enough excess capacity to accommodate any reasonable traffic requirements. Then as new IMPs were added to the system, the capacity was and is still being systematically reduced until the traffic level occupies a substantial fraction of the network's total capacity. At this point, the net's capacity will be increased to maintain the desired percentage of loading. At the initial stages of network design, the "two-connected" reliability constraint essentially determined a minimum value of maximum throughput. This constraint forces the average throughput to be in the range 10-15 kilobits per second per IMP, when 50 kilobit/sec circuits are used throughout the network, since two communication paths between every pair of IMPs are needed. Alternatively, if this level of throughput is required, then the reliability specification of "two-connectivity" can be obtained without additional cost.

Reliability computations

A simple and natural characterization of network reliability is the ability of the network to sustain communication between all operable pairs of IMPs. For design purposes, the requirement of two independent paths between nodes insures that at least two IMPs and/or circuits must fail before any pair of operable IMPs cannot communicate. This criterion is independent of the properties of the IMPs and circuits, does not take into account the "degree" of disruption that may occur and hence, does not reflect the actual availability of resources in the network. A more meaningful measure is the average fraction of IMP pairs that cannct communicate because of IMP and circuit failures. This calculation requires knowledge of the IMP and circuit failure rates, and could not be performed until enough operating data was gathered to make valid predictions.

To calculate network reliability, we must consider elementary network structures known as cutsets. A

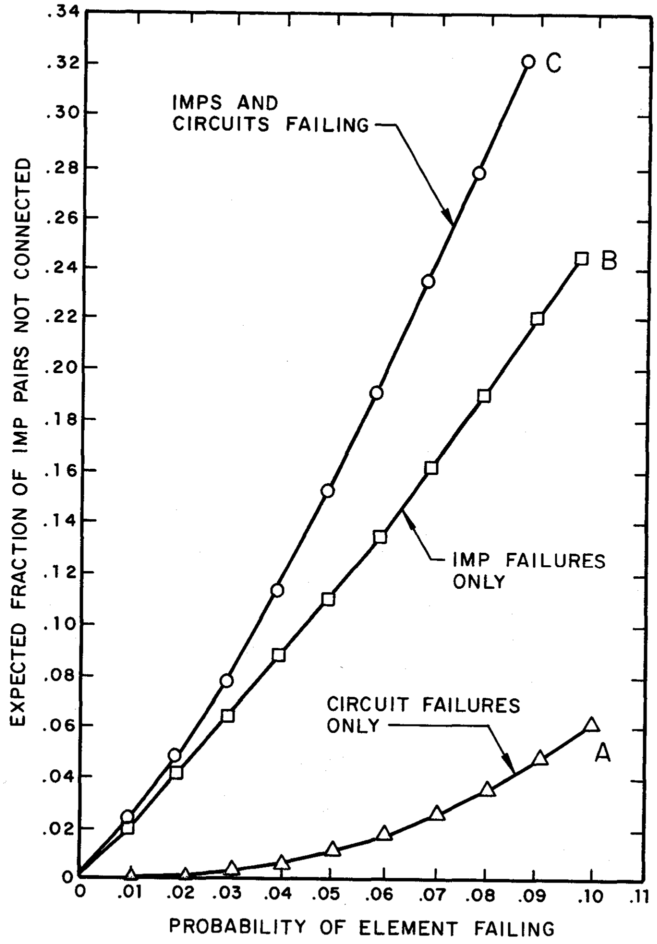


Figure 2—Network availability vs. IMP and circuit reliability

cutset is a set of circuits and/or IMPs whose removal from the network breaks all communication paths between at least two operable IMPs. To calculate reliability, it is often the case that all cutsets must be either enumerated or estimated. As an example, in a 23 IMP, 28 circuit ARPA Network design similar to the one shown in Figure 1(d), there are over twenty million ways of deleting only circuits so that the remaining network has at least one operable pair of IMPs with no intact communication paths. Table 1 indicates the numbers of cutsets in the 23 IMP network as a function of the number of circuits they contain.

A combination of analysis and simulation can be used to compute the average fraction of non-communicating IMP pairs. Detailed descriptions of the analysis methods are given in Reference 44 while their application to the analysis of the ARPANET is discussed in Reference 43. The results of an analysis of the 23 IMP version of the network are shown in Figure 2. The curve marked A shows the results under the assumption that IMPs do not fail, while the curve marked B shows the case where circuits do not fail. The curve marked C assumes that both IMPs and circuits fail with equal probability. In actual operation, the average failure probability of both IMPs and circuits is about 0.02. For this value, it can be seen that the effect of circuit failures is far less significant than the effect of IMP failures. If an IMP fails in a network with n IMPs, at least $n-1$ other IMPs cannot communicate with it. Thus, good network design cannot improve upon the effect directly due to IMP failures, which in the ARPANET is the major factor affecting the reliability of the communications. Further, more intricate reliability analyses which consider the loss of throughput capacity because of circuit failures have also been performed and these losses have been shown to be negligible.[28] Finally, unequal failure rates due to differences in line lengths have been shown to have only minor effects on the analysis and can usually be neglected.[27]

Topological optimization

During the computer optimization process, the reliability of the topology is assumed to be acceptable if the network is at least two-connected. The object of the optimization is to decrease the ratio of cost to throughput subject to an overall cost limitation. This technique employs a sophisticated network optimization program that utilizes circuit exchange heuristics, routing and flow analysis algorithms, to generate low cost designs. In addition, two time delay models were initially used to (1) calculate the throughput corresponding to an average time delay of 0.2 seconds, (2) estimate the packet rejection rate due to all buffers filling at an IMP. As experience with these models grew, the packet rejection rate was found to be negligible and the computation discontinued. The delay computation (Equation (7) in a later section) was subsequently first replaced by a heuristic calculation to speed the computation and later eliminated after it was found that time delays could be guaranteed to be acceptably low by preventing cutsets from being saturated. This "threshold" behavior is discussed further in the next section.

The basic method of optimization was described in Reference 12 while extensions to the design of large networks are discussed in Reference 9. The method operates by initially generating, either manually or by computer, a "starting network" that satisfies the overall network constraints but is not, in general, a low cost network. The computer then iteratively modifies the starting network in simple steps until a lower cost network is found that satisfies the constraints or the process is terminated. The process is repeated until no further improvements can be found. Using a different starting network can result in a different solution. However, by incorporating sensible heuristics and by using a variety of *carefully chosen* starting networks and some degree of man-machine interaction, "excellent" final networks usually result. Experience has shown that there are a wide variety of such networks with different topological structures but similar cost and performance.

The key to this design effort is the heuristic procedure by which the iterative network modifications are made. The method used in the ARPANET design involves the removal and addition of one or two circuits at a time. Many methods have been employed, at various times, to identify the appropriate circuits for potential addition or deletion. For example, to delete uneconomical circuits a straightforward procedure simply deletes single circuits in numerical order, reroutes traffic and reevaluates cost until a decrease in cost per megabit is found. At this point, the deletion is made permanent and the process begins again. A somewhat more sophisticated procedure deletes circuits in order of increasing utilization, while a more complex method attempts to evaluate the effect of the removal of any circuit before any deletion is attempted. The circuit with the greatest likelihood of an improvement is then considered for removal and so on.

There are a huge number of reasonable heuristics for circuit exchanges. After a great deal of experimentation with many of these, it appears that the choice of a particular heuristic is not critical. Instead, the speed and efficiency with which potential exchanges can be investigated appears to be the limiting factor affecting the quality of the final design. Finally, as the size of the network increases, the greater the cost becomes to perform *any* circuit exchange optimization. Decomposition of the network design into regions becomes necessary and additional heuristics are needed to determine effective decompositions. It presently appears that these methods can be used to design relatively efficient networks with a few hundred IMPs while substantially new procedures will be necessary for networks of greater size.

The topological design requires a routing algorithm to evaluate the throughput capability of any given network. Its properties must reflect those of an implementable routing algorithm, for example, within the ARPANET. Although the routing problem can be formulated as a "multicommodity flow problem"[10] and solved by linear programming for networks with 20-30 IMPs,[8] faster techniques are needed when the routing algorithm is incorporated in a design procedure. The design procedure for the ARPA Network topology iteratively analyzes thousands of networks. To satisfy the requirements for speed, an algorithm which selects the least utilized path with the minimum number of IMPs was initially used.[12] This algorithm was later replaced by one which sends as much traffic as possible along such paths until one or more circuits approach a few percent of full utilization.[28] These highly utilized circuits are then no longer allowed to carry additional flow. Instead, new paths with excess capacity and possibly more intermediate nodes are found. The procedure continues until some cutset contains only nearly fully utilized circuits. At this point no additional flow can be sent. For design purposes, this algorithm is a highly satisfactory replacement for the more complicated multi-commodity approach. Using the algorithm, it has been shown that the throughput capabilities of the ARPA Network are substantially insensitive to the distribution of traffic and depend mainly only on the total traffic flow within the network.[8]

Analytic models of network performance

The effort to determine analytic models of system performance has proceeded in two phases: (1) the prediction of average time delay encountered by a message as it passes through the network, and (2) the use of these queueing models to calculate optimum channel capacity assignments for minimum possible delay. The model used as a standard for the average message delay was first described in Reference 21 where it served to predict delays in stochastic communication networks. In Reference 22, it was modified to describe the behavior of ARPA-like computer networks while in Reference 23 it was refined further to apply directly to the ARPANET.

The single server model

Queueing theory[20] provides an effective set of analytical tools for studying packet delay. Much of this theory considers systems in which messages place demands for transmission (service) upon a single communication channel (the single server). These systems are characterized by $A(\tau)$, the distribution of interarrival times between demands and $B(t)$, the distribution of service times. When the average demand for service is less than the capacity of the channel, the system is said to be stable.

When $A(\tau)$ is exponential (i.e., Poisson arrivals), and messages are transmitted on a first-come first-served basis, the average time T in the stable system is

$$T=\frac{\lambda\overline{t^2}}{2(1-\rho)}+\bar{t} \tag{1}$$

where λ is the average arrival rate of messages, $\bar{t}$ and $\overline{t^2}$ are the first and second moments of $B(t)$ respectively, and $\rho=\lambda\bar{t}<1$. If the service time is also exponential,

$$T=\frac{\bar{t}}{1-\rho} \tag{2}$$

When $A(\tau)$ and $B(t)$ are arbitrary distributions, the situation becomes complex and only weak results are available. For example, no expression is available for T; however the following upper bound yields an excellent approximation[19] as $\rho\rightarrow 1$:

$$T\leq\frac{\lambda(\sigma_a{}^2+\sigma_b{}^2)}{2(1-\rho)}+\bar{t} \tag{3}$$

where $\sigma_a{}^2$ and $\sigma_b{}^2$ are the variance of the interarrival time and service time distributions, respectively.

Networks of queues

Multiple channels in a network environment give rise to queueing problems that are far more difficult to solve than single server systems. For example, the variability in the choice of source and destination for a message is a network phenomenon which contributes to delay. A principal analytical difficulty results from the fact that flows throughout the network are correlated. The basic approach to solving these stochastic network problems is to *decompose* them into analyzable single-server problems which reflect the original network structure and traffic flow.

Early studies of queueing networks indicated that such a decomposition was possible;[50,51] however, those results do not carry over to message switched computer networks due to the correlation of traffic flows. In Reference 21 it was shown for a wide variety of communication nets that this correlation could be removed by considering the length of a given packet to be an independent random variable as it passes from node to node. Although this "independence" assumption is not physically realistic, it results in a mathematically tractable model which does not seem to affect the accuracy of the predicted time delays. As the size and connectivity of the network increases, the assumption becomes increasingly more realistic. With this assumption, a successful decomposition which permits a channel-by-channel analysis is possible, as follows.

The packet delay is defined as the time which a packet spends in the network from its entry until it reaches its destination. The average packet delay is denoted as T. Let Z_{jk} be the average delay for those packets whose origin is IMP j and whose destination is IMP k. We assume a Poisson arrival process for such packets with an average of γ_{jk} packets per second and an exponential distribution of packet lengths with an average of $1/\mu$ bits per packet. With these definitions, if γ is the sum of the quantities γ_{jk}, then[21]

$$T=\sum_{j,k}\frac{\gamma_{jk}}{\gamma}Z_{jk} \tag{4}$$

Let us now reformulate Equation (4) in terms of single channel delays. We first define the following quantities for the ith channel: C_i as its capacity (bits/second); λ_i as the average packet traffic it carries (packets/second); and T_i as the average time a packet spends waiting for and using the ith channel. By relating the $\{\lambda_i\}$ to the $\{\gamma_{jk}\}$ via the paths selected by the routing algorithm, it is easy to see that[21]

$$T=\sum_{i}\frac{\lambda_i}{\gamma}T_i \tag{5}$$

With the assumption of Poisson traffic and exponential service times, the quantities T_i are given by Equation (2). For an average packet length of $1/\mu$ bits, $\bar{t}=1/\mu C_i$ seconds and thus

$$T_i=\frac{1}{\mu C_i-\lambda_i} \tag{6}$$

Thus we have successfully decomposed the analysis problem into a set of simple single-channel problems.

A refinement of the decomposition permits a non-exponential packet length distribution and uses Equation (1) rather than Equation (2) to calculate T_i; as an approximation, the Markovian character of the traffic is assumed to be preserved. Furthermore, for computer networks we include the effect of propagation time and overhead traffic to obtain the following equation for average packet delay[22,23]

$$T=K+\sum_{i}\frac{\lambda_i}{\gamma}\left[\frac{1}{\mu' C_i}+\frac{\lambda_i/\mu C_i}{\mu C_i-\lambda_i}+P_i+K\right] \tag{7}$$

Here, $1/\mu'$ represents the average length of a Host packet, and $1/\mu$ represents the average length of all packets (including acknowledgments, headers, requests for next messages, parity checks, etc.) within the network. The expression $1/\mu' C_i+[(\lambda_i/\mu C_i)/(\mu C_i-\lambda_i)]+P_i$ represents the average packet delay on the ith channel. The term $(\lambda_i/\mu C_i)/(\mu C_i-\lambda_i)$ is the average time a packet spends waiting at the IMP for the ith channel to become available. Since the packet must compete with acknowledgments and other overhead traffic, the overall average packet length $1/\mu$ appears in the expression. The term $1/\mu' C_i$ is the time required to transmit a packet of average length μ'. Finally: K is the nodal processing time, assumed constant, and for the ARPA IMP approximately equal to 0.35 ms; P_i is the propagation time on the ith channel (about 20 ms for a 3000 mile channel).

Assuming a relatively homogeneous set of C_i and P_i, no individual term in the expression for delay will dominate the summation until the flow λ_i/μ in one channel (say channel i_o) approaches the capacity C_{i_o}. At that point, the term T_{i_o}, and hence T will grow rapidly. The expression for delay is then dominated by one (or more) terms and exhibits a threshold behavior. Prior to this threshold, T remains relatively constant.

The accuracy of the time delay model, as well as this threshold phenomenon was demonstrated on a 19 node network[14] and on the ten node ARPA net derived from Figure 1(c) by deleting the rightmost five IMPs. Using the routing procedure described in the last section[28] and equal traffic between all node pairs, the channel flows λ_i were found for the ten node net and the delay curves shown in Figure 3 were obtained. Curve A was obtained with fixed 1000 bit packets,* while curve B was generated for exponentially distributed variable length packets with average size of 500 bits. In both cases A and B, all overhead factors were ignored. Note that the delay remains small until a

* In case A, the application of Equation (1) allows for constant packet lengths (i.e., zero variance).

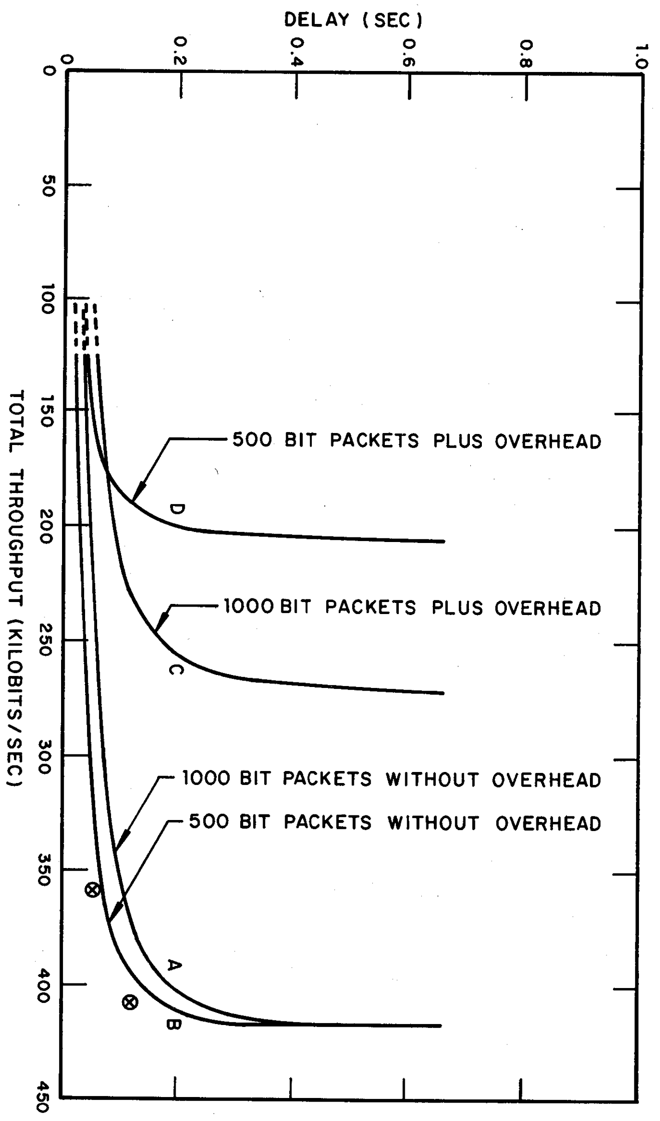


Figure 3—Delay vs. throughput

total throughput slightly greater than 400 kilobits/second is reached. The delay then increases rapidly. Curves *C* and *D* respectively represent the same situations when the overhead of 136 bits per packet and per RFNM and 152 bits per acknowledgment are included. Notice that the total throughput per IMP is reduced to 250 kilobits/second in case *C* and to approximately 200 kilobits/second in case *D*.

In the same figure, we have illustrated with x's the results of a simulation performed with a realistic routing and metering strategy. The simulation omitted all network overhead and assumed fixed lengths of 1000 bits for all packets.

It is difficult to develop a practical routing and flow control procedure that will allow each IMP to input identical amounts of traffic. To compare the delay curve *A* with the points obtained by simulation, the curve should actually be recomputed for the slightly skewed distribution that resulted. It is notable that the delay estimates from the simulation (which used a dynamic routing strategy) and the computation (which used a static routing strategy and the time delay formula) are in close agreement. In particular, they both accurately determined the vertical rise of the delay curve in the range just above 400 kilobits/second, the formula by predicting infinite delay and the simulation by rejecting the further input of traffic.

In practice and from the analytic and simulation studies of the ARPANET, the average queueing delay is observed to remain small (almost that of an unloaded net) and well within the design constraint of 0.2 seconds until the traffic within the network approaches the capacity of a cutset. The delay then increases rapidly. *Thus, as long as traffic is low enough and the routing adaptive enough to avoid the premature saturation of cutsets by guiding traffic along paths with excess capacity, queueing delays are not significant.*

Channel capacity optimization

One of the most difficult design problems is the optimal selection of capacities from a *finite* set of options. Although there are many heuristic approaches to this problem, analytic results are relatively scarce. (For the specialized case of centralized networks, an algorithm yielding optimal results is available.[11]) While it is possible to find an economical assignment of discrete capacities for, say, a 200 IMP network, very little is known about the relation between such capacity assignments, message delay, and cost.

To obtain theoretical properties of optimal capacity assignments, one may ignore the constraint that capacities are obtainable only in discrete sizes. In Reference 21 such a problem was posed where the network topology and average traffic flow were assumed to be known and fixed and an optimal match of capacities to traffic flow was found. Also, the traffic was assumed to be Markovian (Poisson arrivals and exponential packet lengths) and the independence assumption and decomposition method were applied. For each channel, the capacity C_i was found which minimized the average message delay T, at a fixed total system cost D. Since λ_i/μ is the average bit rate on the ith channel, the solution to *any* optimal assignment problem must provide more than this minimal capacity to each channel. This is clear since both Equations (6) and (7) indicate that T_i will become arbitrarily large with less than (or equal to) this amount of capacity. It is not critical exactly how the *excess* capacity is assigned, as long as $C_i > \lambda_i/\mu$. Other important parameters and insights have been identified in studying the continuous capacity optimization problem. For example, the number of excess dollars, D_e, remaining after the minimum capacity λ_i/μ is assigned to each channel is of great importance. As $D_e \rightarrow 0$, the average delay must grow arbitrarily large. In this range, the critical parameters become ρ and $\bar{n}$ where $\rho = \gamma/\mu C$ is the ratio of the rate γ/μ at which bits enter the network to the rate C at which the net can handle bits and $\bar{n} = \lambda/\gamma$, where $\lambda = \sum \lambda_i$ is the total rate at which packets flow within the net. The quantity ρ represents a dimensionless form of network "load" whereas $\bar{n}$ is easily shown to represent the average path length for a packet.

As the load ρ approaches $1/\bar{n}$, the delay T grows very quickly, and this point $\rho = 1/\bar{n}$ represents the maximum load which the network can support. If capacities are assigned optimally, all channels saturate simultaneously at this point. In this formulation $\bar{n}$ is a design parameter which depends upon the topology and the routing procedure, while ρ is a parameter which depends upon the input rate and the total capacity of the network. In studying the ARPANET[23] a closer representation of the actual tariffs for high speed telephone data channels used in that network was provided by setting $D = \sum_i d_i C_i$ where $0 \leq \alpha \leq 1$.* This approach requires the solution of a non-linear equation by numerical techniques. On solving the equation, it can be shown that the packet delay T varies insignificantly with α for $.3 \leq \alpha \leq 1$. This indicates that the closed form solution discussed earlier with $\alpha = 1$ is a reasonable approximation to the more difficult non-linear problem. These continuous capacity studies have application to general network studies (e.g., satellite communications)[33] and are under continued investigation.[25,26,46]

In practice, the selection of channel capacities must be made from a small finite set. Although some theoretical work has been done in this case by approximating the discrete cost-capacity functions by continuous ones, much remains to be done.[13,25] Because of the discrete capacities and the time varying nature of network traffic, it is *not* generally possible to match channel capacities to the anticipated flows within the channels. If this were possible, all channels would saturate at the same externally applied load. Instead, capacities are assigned on the basis of reasonable estimates of average or peak traffic flows. It is the responsibility of the routing procedure to permit the traffic to adapt to the available capacity.[14] Often two IMP sites will engage in heavy communication and thus saturate one or more critical network cutsets. In such cases, the routing will not be able to send additional flow across these cuts. The network will therefore experience "premature" saturation in one or a small set of channels leading to the threshold behavior described earlier.

DISCUSSION

A major conclusion from our experience in network design is that message switched networks of the ARPA type are no longer difficult to specify. They may be implemented straightforwardly from the specifications; they can be less expensive than other currently available technical approaches; they perform remarkably well as a communication system for interconnecting time-sharing and batch processing computers and can be adapted to directly handle teletypes, displays and many other kinds of terminal devices and data processing equipment.[16,30]

The principal tools available for the design of networks are analysis, simulation, heuristic procedures, and experimentation. Analysis, simulation and heuristics have been the mainstays of the work on modeling and topological optimization while simulation, heuristic procedures and experimental techniques have been the major tools for the actual network implementation. Experience has shown that all of these methods are useful while none are all powerful. The most valuable approach has been the simultaneous use of several of these tools.

Each approach has room for considerable improvement. The analysis efforts have not yet yielded results in many important areas such as routing. However, for prediction of delay, this approach leads to a simple threshold model which is both accurate and understandable. Heuristic procedures all suffer from the problem that it is presently unclear how to select appropriate heuristics. It has been the innovative use of computers and analysis that has made the approach work well. For designing networks with no more than a few hundred IMPs, present heuristics appear adequate but a good deal of additional work is required for networks of greater size. Simulation is a well developed tool that is both expensive to apply and limited in the overall understanding that it yields. For these reasons, simulation appears to be most useful only in validating models, and in assisting in detailed design decisions such as the number of buffers that an IMP should contain. As the size of networks continues to grow, it appears that simulation will become virtually useless as a total design tool. The ultimate standard by which all models and

* Of course the tariffs reflect the discrete nature of available channels. The use of the exponent α provides a continuous fit to the discrete cost function. For the ARPANET, $\alpha \cong .8$.

conclusions can be tested is experimentation. Experimentation with the actual network is conceptually relatively straightforward and very useful. Although, experiments are often logistically difficult to perform, they can provide an easy means for testing models, heuristics and design parameters.

The outstanding design problems currently facing the network designer are to specify and determine the properties of the routing, flow control and topological structure for large networks. This specification must make full use of a wide variety of circuit options. Preliminary studies indicate that initially, the most fruitful approaches will be based on the partitioning of the network into regions, or equivalently, constructing a large network by connecting a number of regional networks. To send a message, a Host would specify both the destination region and the destination IMP in that region. No detailed implementation of a large network has yet been specified but early studies of their properties indicate that factors such as cost, throughput, delay and reliability are similar to those of the present ARPANET, if the ARPA technology is used.[9]

Techniques applicable to the design of large networks are presently under intensive study. These techniques appear to split into the same four categories as small network design but approaches may differ significantly. For example, large nets are likely to demand the placement of high bandwidth circuits at certain key locations in the topology to concentrate flow. These circuits will require the development of a high speed IMP to connect them into the net. It is likely that this high speed IMP will have the structure of a high speed multiplexor, and may require several cooperating processors to obtain the needed computer power for the job. Flow control strategies for large networks seem to extrapolate nicely from small network strategies if each region in the large network is viewed as a node in a smaller network. However, this area will require additional study as will the problem of specifying effective adaptive routing mechanisms. Recent efforts indicate that efficient practical schemes for small networks will soon be available. These schemes seem to be applicable for adaptive routing and flow control in networks constructed from regional subnetworks. The development of practical algorithms to handle routing and flow control is still an art rather than a science. Simulation is useful for studying the properties of a given heuristic, but intuition still plays a dominant role in the system design.

Several open questions in network design presently are: (1) what structure should a high bandwidth IMP have; (2) how can full use be made of a variety of high bandwidth circuits; (3) how should large networks be partitioned for both effective design and operation; and (4) what operational procedures should large networks follow? Much work has already been done in these areas but much more remains to be done. We expect substantial progress to be achieved in the next few years, and accordingly, the increased understanding of the properties of message switched networks of all sizes.

ACKNOWLEDGMENT

The ARPA Network is in large part the conception of Dr. L. G. Roberts of the Advanced Research Projects Agency to whom we owe a debt of gratitude for his support and encouragement. We also acknowledge the helpful contributions of S. Crocker and B. Dolan of ARPA. At BBN, NAC, and UCLA many individuals, too numerous to list, participated in the network effort and we gratefully acknowledge their contributions.

REFERENCES

1 P BARAN S BOEHM P SMITH
On distributed communications
Series of 11 reports by Rand Corporation Santa Monica California 1964

2 "*Specifications for the interconnection of a Host and an IMP*
BBN Report No 1822 1971 revision

3 S CARR S CROCKER V CERF
Host-Host communication protocol in the ARPA network
SJCC 1970 pp 589-597

4 S CROCKER et al
Function oriented protocols for the ARPA network
SJCC 1972 in this issue

5 D W DAVIES
The control of congestion in packet switching networks
Proc of the Second ACM IEEE Symposium on problems in the Optimization of Data Communications Systems Palo Alto California Oct 1971 pp 46-49

6 D FARBER K LARSON
The architecture of a distributed computer system—An informal description
University of California Irvine Information and Computer Science Technical Report #11 1970

7 W D FARMER E E NEWHALL
An experimental distribution switching system to handle bursty computer traffic
Proc of the ACM Symposium on Problems in the Optimization of Data Communication Systems 1969 pp 1-34

8 H FRANK W CHOU
Routing in computer networks
Networks John Wiley 1971 Vol 1 No 2 pp 99-112

9 H FRANK W CHOU
Cost and throughput in computer-communication networks
To appear in the Infotech Report on the State of the Art of Computer Networks 1972

10 H FRANK I T FRISCH
Communication transmission and transportation networks
Addison Wesley 1972

11 H FRANK I T FRISCH W CHOU
R VAN SLYKE
Optimal design of centralized computer networks
Networks John Wiley Vol 1 No 1 pp 43-57 1971

12 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
SJCC May 1970 pp 581-587

13 L FRATTA M GERLA L KLEINROCK
The flow deviation method; An approach to store-and-forward network design
To be published

14 G FULTZ L KLEINROCK
Adaptive routing techniques for store-and-forward computer-communication networks
Proc of the International Conference on communications 1971 pp 39-1 to 39-8

15 F HEART R KAHN S ORNSTEIN
W CROWTHER D WALDEN
The interface message processor for the ARPA computer network
SJCC 1970 pp 551-567

16 R E KAHN
Terminal access to the ARPA computer network
Proc of Third Courant Institute Symposium Nov 1970
To be published by Prentice Hall Englewood Cliffs, NJ

17 R E KAHN W R CROWTHER
Flow control in a resource sharing computer network
Proc of the Second ACM IEEE Symposium on Problems in the Optimization of Data Communications Systems Palo Alto California October 1971 pp 108-116

18 R E KAHN W R CROWTHER
A study of the ARPA network design and performance
BBN Report No 2161 August 1971

19 J F C KINGMAN
Some inequalities for the queue GI/G/1
Biometrica 1962 pp 315-324

20 L KLEINROCK
Queueing systems; Theory and application
To be published by John Wiley 1972

21 L KLEINROCK
Communication nets: Stochastic message flow and delay
McGraw-Hill 1964

22 L KLEINROCK
Models for computer networks
Proc of the International Conference on Communications 1969 pp 21-9 to 21-16

23 L KLEINROCK
Analysis and simulation methods in computer network design
SJCC 1970 pp 569-579

24 T MARILL L G ROBERTS
Toward a cooperative network of time-shared computers
FJCC 1966

25 B MEISTER H MULLER H RUDIN JR
Optimization of a new model for message-switching networks
Proc of the International Conference on Communications 1971 pp 39-16 to 39-21

26 B MEISTER H MULLER H RUDIN
New optimization criteria for message-switching networks
IEEE Transactions on Communication Technology Com-19 June 1971 pp 256-260

27 N. A. C. Third Semiannual Technical Report for the Project Analysis and Optimization of Store-and-Forward Computer Networks
Defense Documentation Center Alexandria Va
June 1971

28 N. A. C. Fourth Semiannual Technical Report for the Project Analysis and Optimication of Store-and-Forward Computer Networks
Defense Documentation Center Alexandria Va Dec 1971

29 *The Host/Host protocol for the ARPA network*
Network Working Group N I C No 7147 1971 (Available from the Network Information Center Stanford Research Institute Menlo Park California)

30 ORNSTEIN et al
The terminal IMP for the ARPA network
SJCC 1972 In this issue

31 J PIERCE
A network for block switching of data
IEEE Convention Record New York N Y March 1971

32 E PORT F CLOS
Comparisons of switched data networks on the basis of waiting times
IBM Research Report RZ 405 IBM Research Laboratories Zurich Switzerland Jan 1971 (Copies available from IBM Watson Research Center
P O Box 218 Yorktown Heights New York 10598)

33 H G RAYMOND
A queueing theory approach to communication satellite network design
Proc of the International Conference on Communication pp 42-26 to 42-31 1971

34 L G ROBERTS
Multiple computer networks and inter-computer communications
ACM Symposium on Operating Systems
Gatlinburg Tenn 1967

35 L G ROBERTS
A forward look
SIGNAL Vol XXV No 12 pp 77-81 August 1971

36 L G ROBERTS
Resource sharing networks
IEEE International Conference March 1969

37 L G ROBERTS
Access control and file directories in computer networks
IEEE International Convention March 1968

38 L G ROBERTS B WESSLER
Computer network development to achieve resource sharing
SJCC 1970 pp 543-549

39 L G ROBERTS B WESSLER
The ARPA computer network
In Computer Communication Networks edited by Abramson and Kuo Prentice Hall 1972

40 R A SCANTLEBURY P T WILKINSON
The design of a switching system to allow remote access to computer services by other computers
Second ACM/IEEE Symposium on Problems in the Optimization of Data Communications Systems
Palo Alto California October 1971

41 R A SCANTLEBURY
A model for the local area of a data communication network—Objectives and hardware organization
ACM Symposium on Problems in the Optimization of Data Communication Systems Pine Mountain Ga 1969 pp 179-201

42 R H THOMAS D A HENDERSON
McRoss—A multi-computer programming system
SJCC 1972 In this issue

43 R VAN SLYKE H FRANK
Reliability of computer-communication networks
Proc of the Fifth Conference on Applications of Simulation New York December 1971

44 R VAN SLYKE H FRANK
Network reliability analysis—I
Networks Vol 1 No 3 1972

45 E WOLMAN
A fixed optimum cell size for records of various length
JACM 1965 pp 53-70

46 L KLEINROCK
Scheduling, queueing and delays in time-sharing systems and computer networks
To appear in Computer-Communication Networks edited by Abramson and Kuo Prentice Hall 1972

47 A H WEIS
Distributed network activity at IBM
IBM Research Report RC3392 June 1971

48 M BEERE N SULLIVAN
Tymnet—A serendipitous evolution
Second ACM IEEE Symposium on Problems in the Optimization of Data Communications Systems Palo Alto California 1971 pp 16-20

49 W TEITELMAN R E KAHN
A network simulation and display program
Third Princeton Conference on Information Sciences and Systems 1969 p 29

50 P BURKE
The output of a queueing system
Operations Research 1956 pp 699-704

51 J R Jackson
Networks of waiting lines
Operations Research Vol 5 1957 pp 518-521

52 D B McKAY D P KARP
Network/440—IBM Research Computer Sciences Department Computer Network
IBM Research Report RC3431 July 1971

Function-oriented protocols for the ARPA Computer Network

by STEPHEN D. CROCKER
Advanced Research Projects Agency
Arlington, Virginia

JOHN F. HEAFNER
The RAND Corporation
Santa Mon ca, California

ROBERT M. METCALFE
Massachusetts Institute of Technology
Cambridge, Massachusetts

and

JONATHAN B. POSTEL
University of California
Los Angeles, California

INTRODUCTION

Much has been said about the mechanics of the ARPA Computer Network (ARPANET) and especially about the organization of its communications subnet.[1,2,3,4,5] Until recently the main effort has gone into the implementation of an ARPANET user-level communications interface. Operating just above the communications subnet in ARPANET HOST Computers, this ARPANET interface is intended to serve as a foundation for the organization of function-oriented communications.[6,7] See Figures 1 and 2 for our view of a computer system and the scheme for user-level process-to-process communications. It is now appropriate to review the development of protocols which have been constructed to promote particular substantive uses of the ARPANET, namely function-oriented protocols.

We should begin this brief examination by stating what we mean by the word "protocol" and how protocols fit in the plan for useful work on the ARPANET. When we have two processes facing each other across some communication link, the protocol is the set of their agreements on the format and relative timing of messages to be exchanged. When we speak of a protocol, there is usually an important goal to be fulfilled. Although any set of agreements between cooperating (i.e., communicating) processes is a protocol, the protocols of interest are those which are constructed for general application by a large population of processes in solving a large class of problems.

In the understanding and generation of protocols there are two kinds of distinctions made. Protocols in the ARPANET are *layered* and we speak of high or low level protocols. High level protocols are those most closely matched to functions and low level protocols deal with communications mechanics. The lowest level software protocols in the ARPANET involve reliable message exchange between ARPANET Interface Message Processors (IMPs).[2,5] A high level protocol is one with primitives closely related to a substantive use. At the lowest levels the contents of messages are unspecified. At higher levels, more and more is stated about the meaning of message contents and timing. The layers of protocol are shown in Figure 3.

A second way of structuring sets of protocols and their design is bound up in the word *factoring.* At any level of protocol are sets of format and timing rules associated with particular groupings of agreements. In the IMPs we find certain protocols pertaining to error handling, while others to flow control, and still others to message routing. At the ARPANET's user-level communications interface there are, among others, separable protocols associated with establishing connections and logical data blocking. These protocols do not nest, but join as modules at the same level.

Before moving on to consider the higher level function-oriented protocols, let us first make a few statements about underlying protocols. There are three lower level software protocols which nest in support of the user-level communications interface for the ARPANET. The lowest of these is the IMP-IMP protocol which provides for reliable communication among IMPs. This protocol handles transmission-error detection and correction, flow control to avoid message congestion, and routing. At the next higher level is the IMP-HOST protocol which provides for the passage of messages between HOSTs and IMPs in such a way as to create virtual communication paths between HOSTs. With IMP-HOST protocol, a HOST has operating rules which permit it to send messages to specified HOSTs on the ARPANET and to be informed of the dispensation of those messages. In particular, the IMP-HOST protocol constrains HOSTs in their transmissions so that they can make good use of available

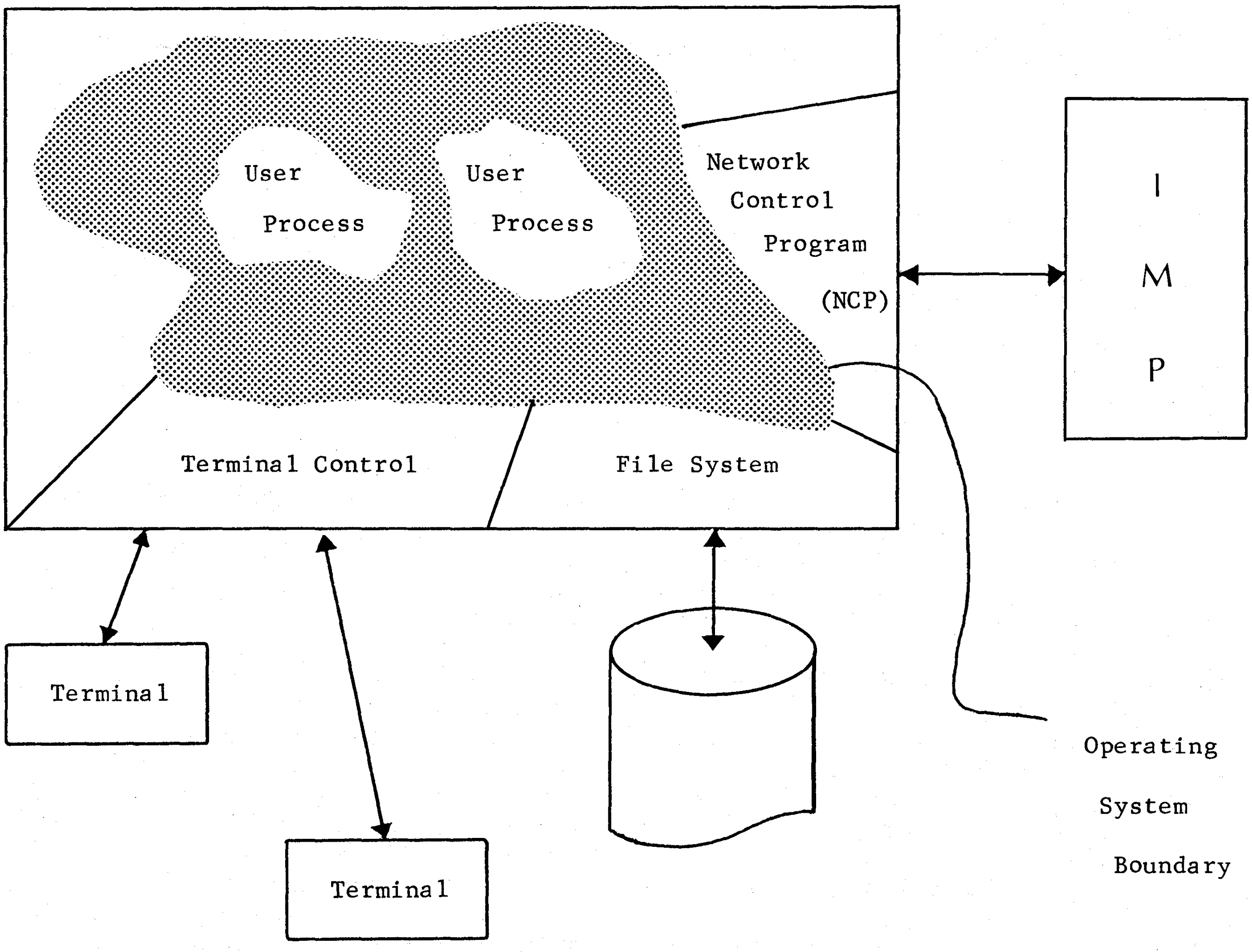


Figure 1—Our view of a computer system

communications capacity without denying such availability to other HOSTs.

The HOST-HOST protocol, finally, is the set of rules whereby HOSTs construct and maintain communication between processes (user jobs) running on remote computers. One process requiring communications with another on some remote computer system makes requests on its local supervisor to act on its behalf in establishing and maintaining those communications under HOST-HOST protocol.

In constructing these low levels of protocol it was the intention to provide user processes with a general set of useful communication primitives to isolate them from many of the details of operating systems and communications. At this user-level interface function-oriented protocols join as an open-ended collection of modules to make use of ARPANET capabilities.

The communications environment facing the designers of function-oriented protocols in the ARPANET is essentially that of a system of one-way byte-oriented connections. Technically speaking, a "connection" is a pair: a "send socket" at one end and a "receive socket" at the other. Primitives provided at the user-level interface include:

1. Initiate connection (local socket, foreign socket),
2. Wait for connection (local socket),
3. Send, Receive (local socket, data),
4. Close (local socket),
5. Send interrupt signal (local socket).

Processes in this virtual process network can create connections and transmit bytes. Connections are subject to HOST-HOST flow control and the vagaries of timing in a widely distributed computing environment, but care has been taken to give user processes control over their communications so as to make full use of network parallelism and redundancy. The kind of agreements which must be made in the creation of

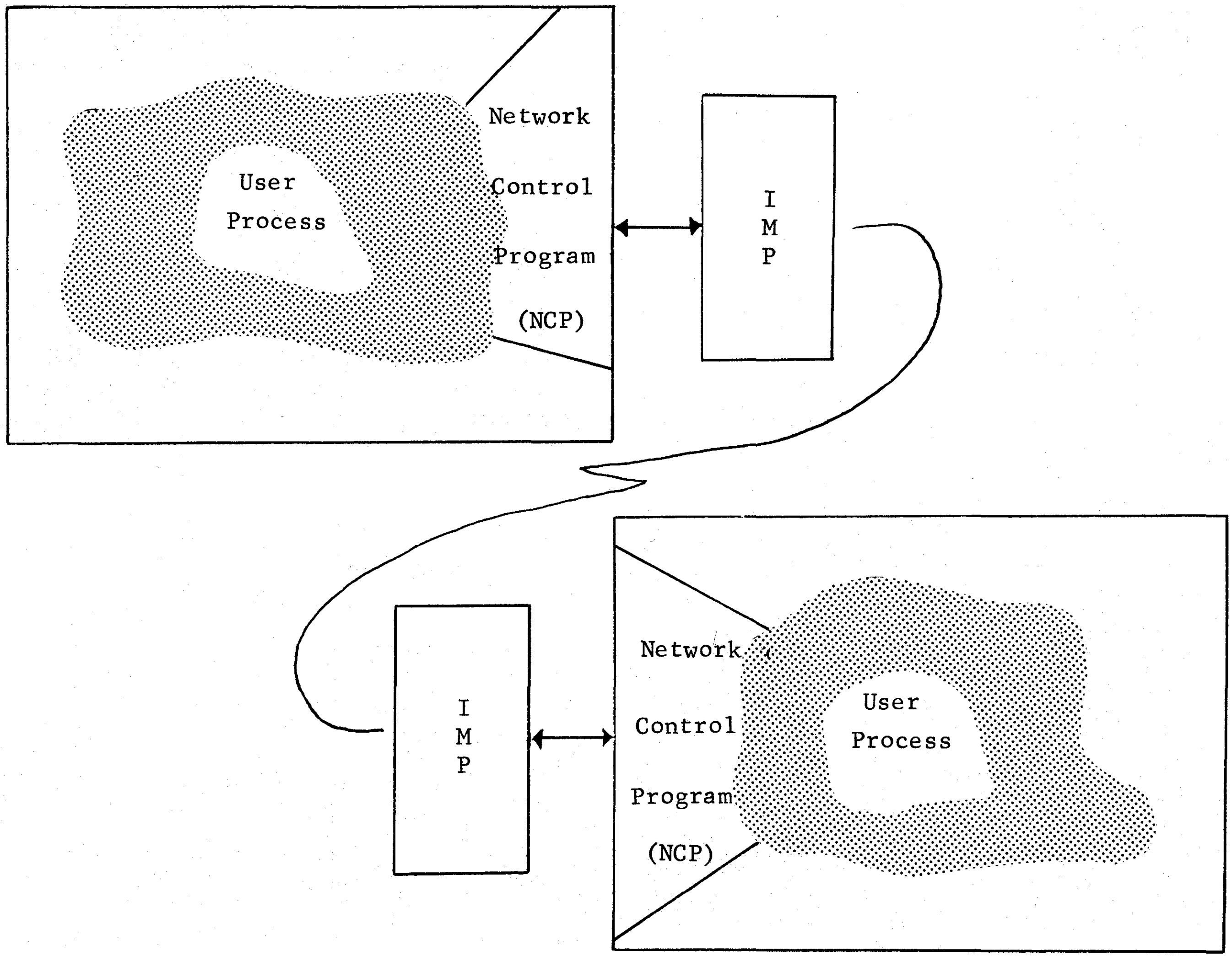


Figure 2—Two communicating processes

function-oriented protocols relate to rules for establishing connections, to the timing rules which govern transmission sequences, and to the content of the byte-streams themselves.

USE OF REMOTE INTERACTIVE SYSTEMS

The application which currently dominates ARPANET activity is the remote use of interactive systems. A Telecommunications Network (TELNET) protocol is followed by processes cooperating to support this application.[8] A user at a terminal, connected to his local HOST, controls a process in a remote HOST as if he were a local user of the remote HOST. His local HOST copies characters between his terminal and TELNET connections over the ARPANET. We refer to the HOST where the user sits as the *using HOST*, and to the remote HOST as the *serving HOST*. See Figure 4.

At the using HOST, the user must be able to perform the following functions through his TELNET user process ("user-TELNET"):

1. Initiate a pair of connections to a serving HOST,
2. Send characters to the serving HOST,
3. Receive characters from the serving HOST,
4. Send a HOST-HOST interrupt signal,
5. Terminate connections.

The user-TELNET needs to be able to distinguish between (1) commands to be acted on locally and (2) input intended for the serving HOST. An escape character is reserved to mark local commands. Conventions for the ARPANET Terminal IMP (TIP) user-TELNET are typical.[9]

In most using HOSTs, the above functions are provided by a user-TELNET which is a *user-level program*. A minimal user-TELNET need only implement the above functions, but several additional support func-

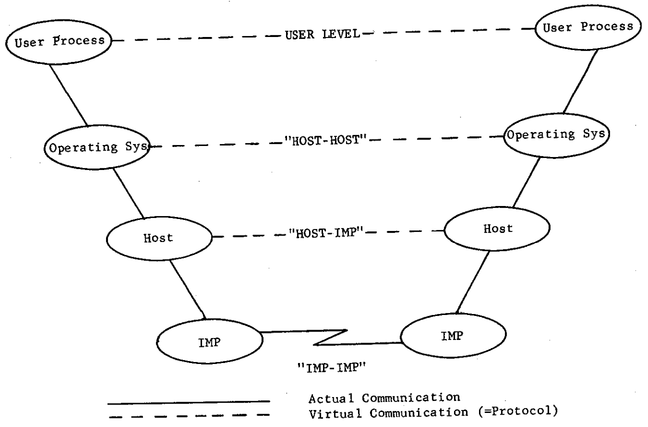


Figure 3—The layers of protocol

tions are often provided (e.g., saving a transcript of a session in a local file, sending a file in place of user-typed input, reporting whether various HOSTs are or have been up).

In the serving HOST it is desirable that a process controlled over the ARPANET behave as it would if controlled locally. The cleanest way to achieve this goal is to generalize the terminal control portion (TCP) of the operating system to accept ARPANET terminal interaction. It is unpleasant to modify any portion of a working computer system and modification could be avoided if it were possible to use a non-supervisor process (e.g., "server-TELNET" or "LOGGER") to perform the job creation, login, terminal input-output, interrupt, and logout functions in exactly the same way

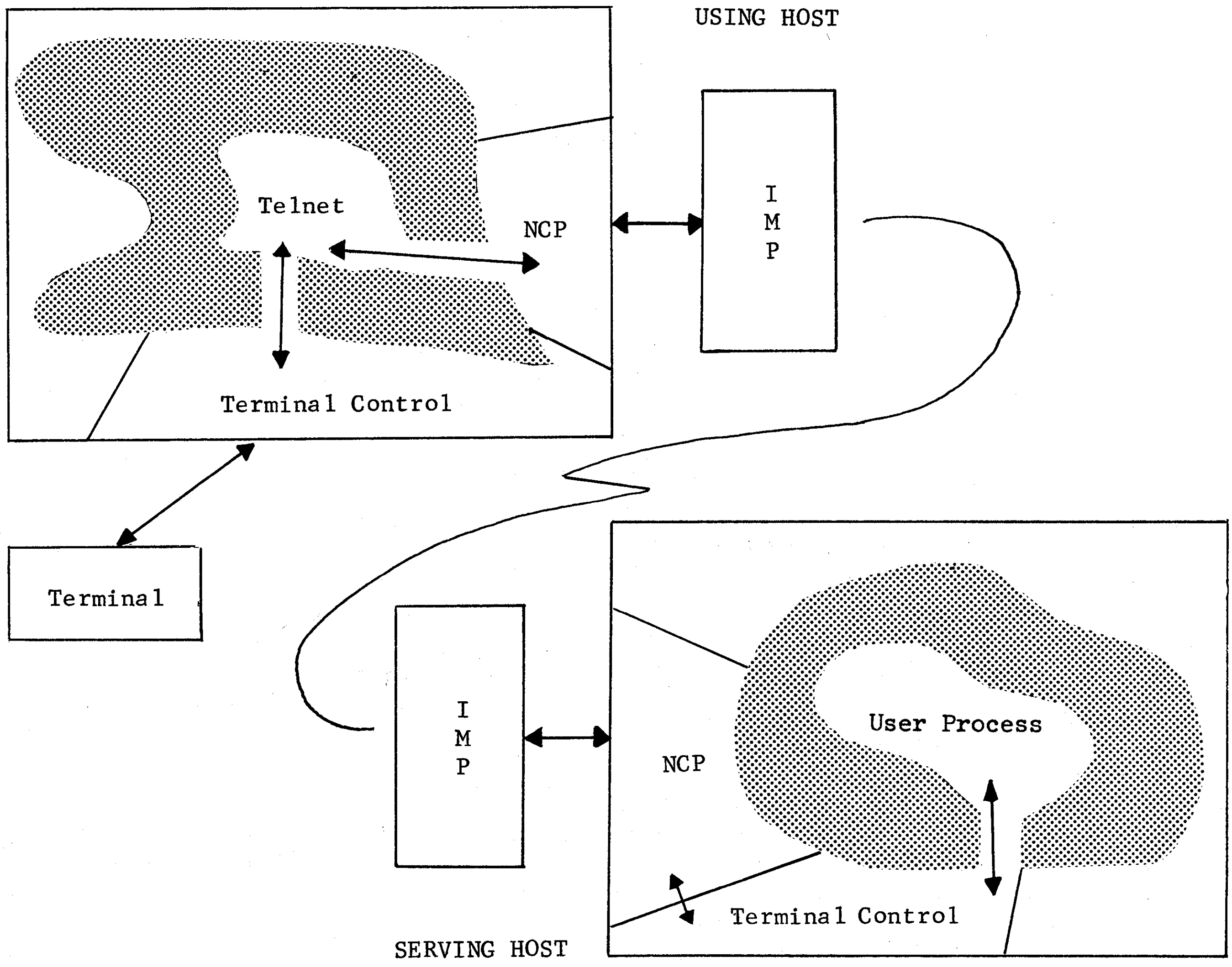


Figure 4—Data flow for remote interactive use

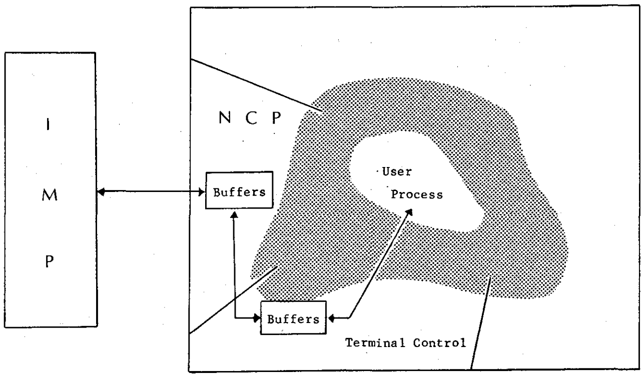


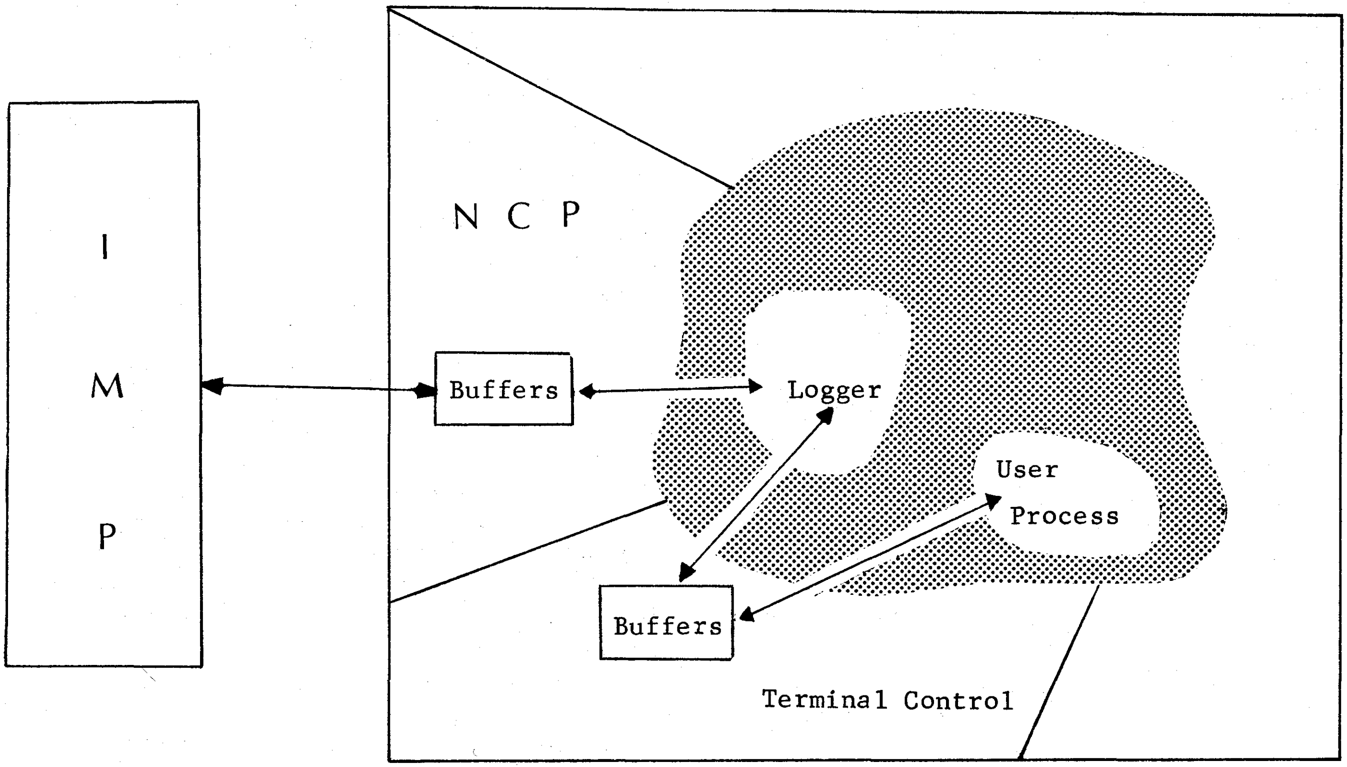
Figure 5—Data flow scheme for server

as a direct console user. Prior to the development of the ARPANET, no operating system provided these functions to non-supervisor processes in anywhere near the required completeness. Some systems have since been modified to support this generalized job control scheme. See Figures 5 and 6.

Efforts to standardize communications in the TELNET protocol focused on four issues: character set, echoing, establishing connections, and attention handling.

The chosen character set is 7-bit ASCII in 8-bit fields with the high-order bit off. Codes with the high-order bit on are reserved for TELNET control functions. Two such TELNET control function codes are the "long-space" which stands for the 200 millisecond space generated by the teletype BREAK button, and the synchronization character (SYNCH) discussed below in conjunction with the purpose of the TELNET interrupt signal.

Much controversy existed regarding echoing. The basic problem is that some systems expect to echo, while some terminals always echo locally. A set of conventions and signals was developed to control which side of a TELNET connection should echo. In practice, those systems which echo have been modified to include provision for locally echoing terminals. This is a nontrivial change affecting many parts of a serving HOST. For example, normally echoing server HOSTs do not echo passwords so as to help maintain their security. Terminals which echo locally defeat this strategy, how-



LOGGER must be a background service program capable of initiating jobs

Figure 6—Alternate data flow scheme for a server

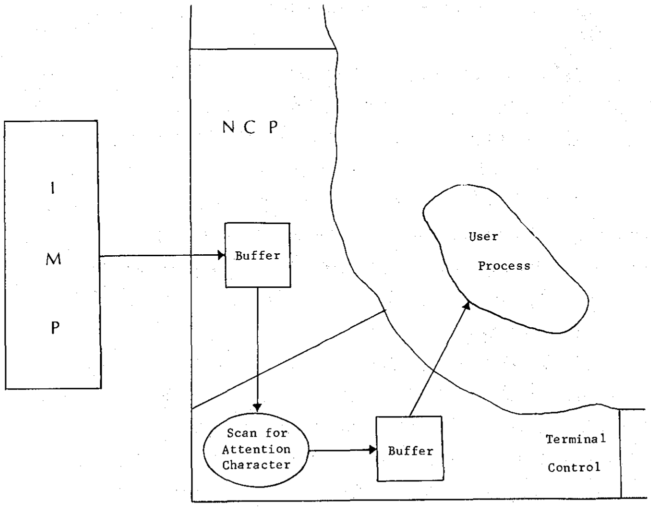


Figure 7—Data flow and processing of the character input stream

ever, and some other protection scheme is necessary. Covering the password with noise characters is the usual solution.

The HOST-HOST protocol provides a large number of sockets for each HOST, but carefully refrains from specifying which ones are to be used for what. To establish communication between a user-TELNET and a server-TELNET some convention is required. The Initial Connection Protocol (ICP)[10] is used:

1. Connection is initiated from a user-TELNET's receive socket to a serving HOST's socket 1 (a send socket).
2. When the initial connection is established, the serving HOST sends a generated socket number and closes the connection. This socket number identifies an adjacent socket pair at the serving HOST through which the user-TELNET can communicate with a server-TELNET.
3. TELNET connections are then initiated between the now specified pairs of sockets. Two connections are used to provide bi-directional communication.

Note that socket 1 at the serving HOST is in use only long enough to send another socket number with which to make the actual service connections.

One of the functions performed by a terminal control program within an operating system is the scanning of an input stream for attention characters intended to stop an errant process and to return control to the executive. Terminal control programs which buffer input sometimes run out of space. When this happens to a local terminal's input stream, a "bell" or a question mark is echoed and the overflow character discarded, *after checking to see if it is the attention character.* See Figure 7. This strategy works well in practice, but it depends rather strongly on the intelligence of the human user, the invariant time delay in the input transmission system, and a lack of buffering between type-in and attention checking. None of these conditions exists for interactive traffic over the net: The serving HOST cannot control the speed (except to slow it down) or the buffering within the using HOST, nor can it even know whether a human user is supplying the input. It is thus necessary that the terminal control program or server-TELNET not, in general, discard characters from a network input stream; instead it must suspend its acceptance of characters via the HOST-HOST flow control mechanism. Since a HOST may only send messages when there is room at the destination, the responsibility for dealing with too much input is thus transferred back to the using HOST. This scheme assures that no characters accepted by the using HOST are inadvertently lost. However, if the process in the serving HOST stops accepting input, the pipeline of buffers between the user-TELNET and remote process will fill up so that attention characters cannot get through to the serving executive. In the TELNET protocol, the solution to this problem calls for the user-TELNET to send, on request, a HOST-HOST interrupt signal forcing the server-TELNET to switch input modes to process network input for attention characters. The server-TELNET is required to scan for attention characters in its network input, even if some input must be discarded while doing so. The effect of the interrupt signal to a server-TELNET from its user is to cause the buffers between them to be emptied for the priority processing of attention characters.

To flip an attention scanning server-TELNET back into its normal mode, a special TELNET synchronization character (SYNCH) is defined. When the server-TELNET encounters this character, it returns to the strategy of accepting terminal input only as buffer space permits. There is a possible race condition if the SYNCH character arrives before the HOST-HOST interrupt signal, but the race is handled by keeping a count of SYNCHs without matching signals. Note that attention characters are HOST specific and may be any of 129 characters—128 ASCII plus "long space"—while SYNCH is a TELNET control character recognized by all server-TELNETs. It would not do to use the HOST-HOST signal alone in place of the signal-SYNCH combination in attention processing, because the position of the SYNCH character in the TELNET input stream is required to determine where attention processing ends and where normal mode input processing begins.

FILE TRANSFER

When viewing the ARPANET as a distributed computer operating system, one initial question is that of how to construct a distributed file system. Although it is constructive to entertain speculation on how the ultimate, automated distributed file system might look, one important first step is to provide for the simplest kinds of explicit file transfer to support early substantive use.

During and immediately after the construction of the ARPANET user-level process interface, several *ad hoc* file transfer mechanisms developed to provide support for initial use. These mechanisms took two forms: (1) use of the TELNET data paths for text file transfer and (2) use of raw byte-stream communication between compatible systems.

By adding two simple features to the user-TELNET, text file transfer became an immediate reality. By adding a "SCRIPT" feature to user-TELNETS whereby all text typed on the user's console can be directed to a file on the user's local file system, a user need only request of a remote HOST that a particular text file be typed on his console to get that file transferred to his local file system. By adding a "SEND-FILE" feature to a user-TELNET whereby the contents of a text file can be substituted for console type-in, a user need only start a remote system's editor as if to enter new text and then send his local file as type-in to get it transferred to the remote file system. Though crude, both of these mechanisms have been used with much success in getting real work done.

Between two identical systems it has been a simple matter to produce programs at two ends of a connection to copy raw bits from one file system to another. This mechanism has also served well in the absence of a more general and powerful file transfer system.

Ways in which these early *ad hoc* file transfer mechanisms are deficient are that (1) they require explicit and often elaborate user intervention and (2) they depend a great deal on the compatibility of the file systems involved. There is an on-going effort to construct a File Transfer Protocol (FTP)[11,12] worthy of wide implementation which will make it possible to exchange structured sequential files among widely differing file systems with a minimum (if any) explicit user intervention. In short, the file transfer protocol being developed provides for the connection of a file transfer user process ("user-FTP") and file transfer server process ("server-FTP") according to the Initial Connection Protocol discussed above. See Figure 8. A user will be able to request that specific file manipulation operations be performed on his behalf. The File Transfer Protocol will support file operations including (1) list remote directory, (2) send local file, (3) retrieve remote file, (4) rename remote file, and (5) delete remote file.

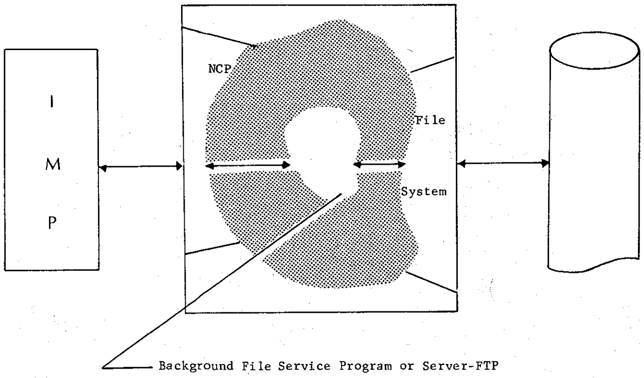


Figure 8—Data flow for file transfer

It is the intention of the protocol designers to regularize the protocol so that file transfer commands can be exchanged by consoles file transfer jobs engaged in such exotic activities as automatic back-up and dynamic file migration. The transfers envisioned will be accompanied with a Data Transfer Protocol (DTP)[11] rich enough to preserve sequential file structure and in a general enough way to permit data to flow between different file systems.

USING THE ARPANET FOR REMOTE JOB ENTRY

A very important use of the ARPANET is to give a wide community of users access to specialized facilities. One type of facility of interest is that of a very powerful number-cruncher. Users in the distributed ARPANET community need to have access to powerful machines for compute-intensive applications and the mode of operation most suited to these uses has been batch Remote Job Entry (RJE). Typically, a user will generate a "deck" for submission to a batch system. See Figure 9. He expects to wait for a period on the order of tens of minutes or hours for that "deck" to be processed, and then to receive the usually voluminous output thereby generated. See Figure 10.

As in the case of file transfer, there are a few useful *ad hoc* ARPANET RJE protocols. A standard RJE protocol is being developed to provide for job submission to a number of facilities in the ARPANET. This protocol is being constructed using the TELNET and File Transfer protocols. A scenario which sketches how the protocol provides the RJE in the simplest, most explicit way is as follows:

Via an ARPANET RJE process, a user connects his

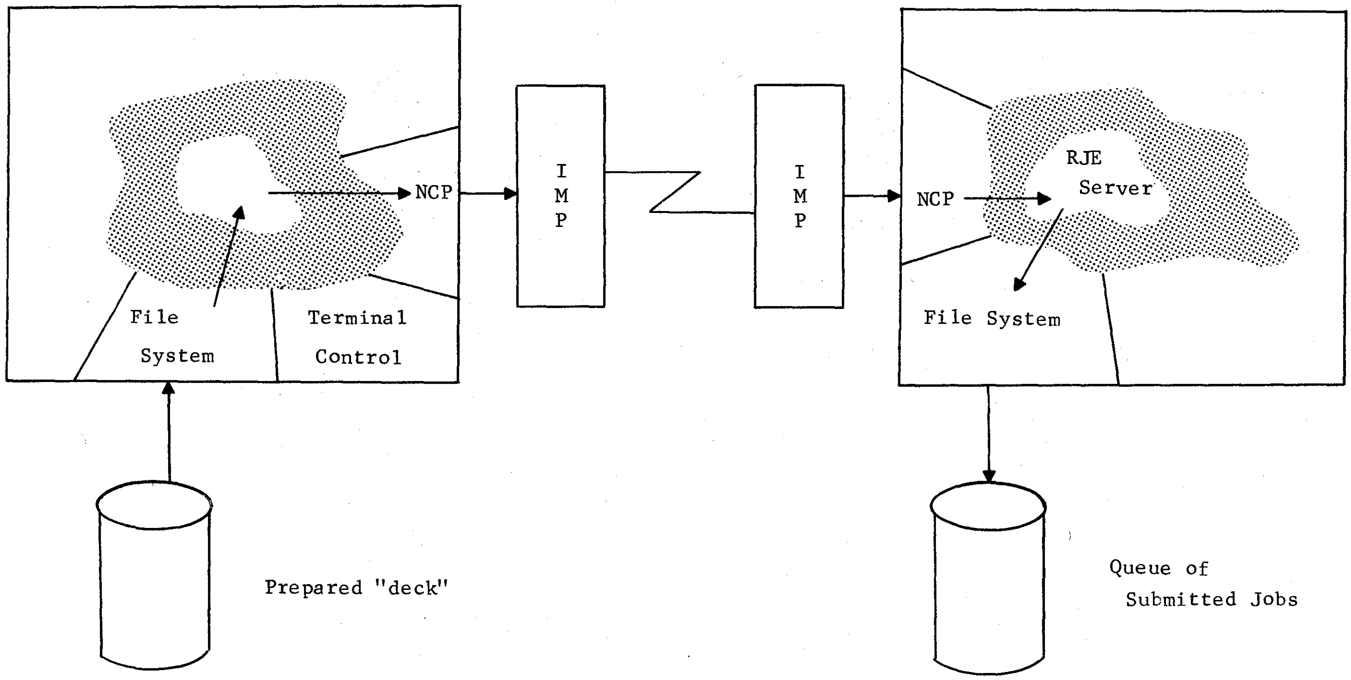


Figure 9—Submission of RJE input

terminal to an RJE server process at the HOST to which he intends to submit his job "deck." Through a short dialogue, he establishes the source of his input and initiates its transfer using the File Transfer Protocol. At some later time, the user reconnects to the appropriate RJE server and makes an inquiry on the

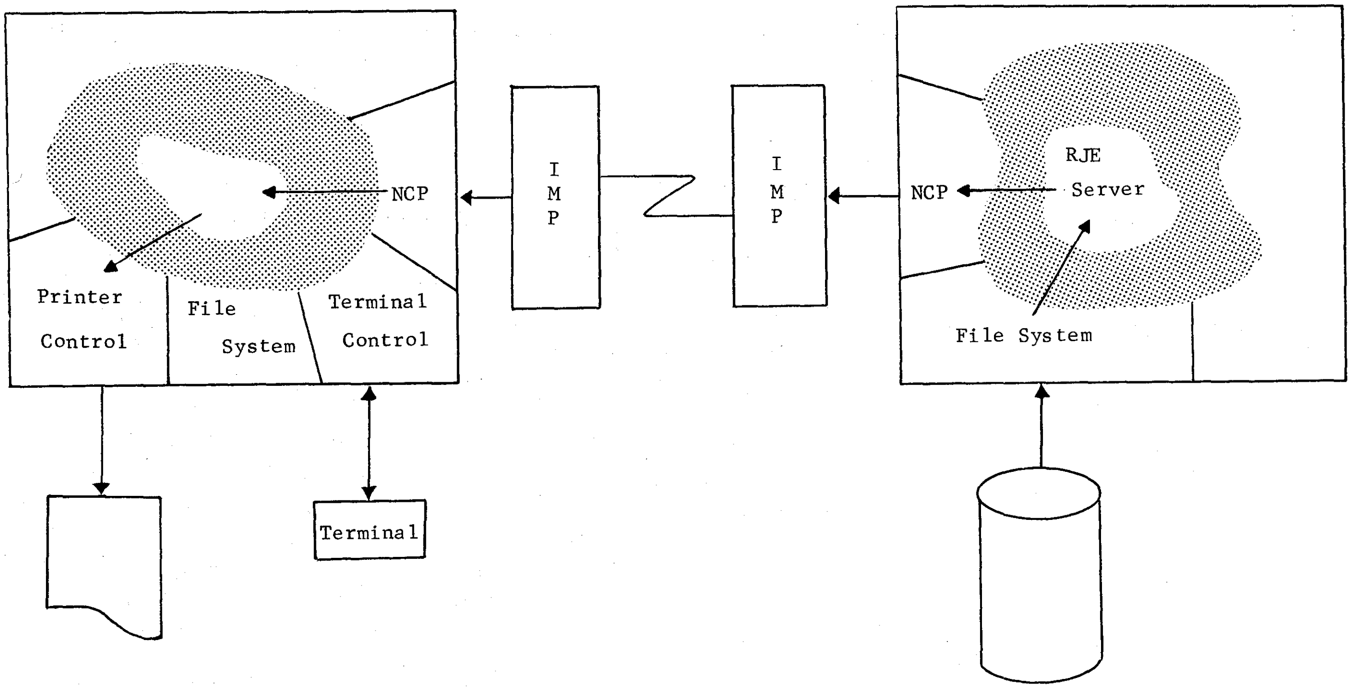


Figure 10—Retrieval of RJE output

status of his job. When notified that his input has been processed, he then issues commands to the serving HOST to transfer his output back.

We can of course imagine more automatic ways of achieving these same functions. A user might need only type a job submission command to his local system. Automatically and invisibly, then, the local system would connect and converse with the specified RJE server causing the desired output to later appear in the users file area or perhaps on a local line printer. The intention is to design the RJE protocol so that the explicit use can start immediately and the more automatic RJE systems can be built as desired.

OTHER PROTOCOLS AND CONCLUSIONS

One of the more difficult problems in utilizing a network of diverse computers and operating systems is that of dealing with incompatible data streams. Computers and their language processors have many ways of representing data. To make use of different computers it is necessary to (1) produce a mediation scheme for each incompatibility or (2) produce a standard representation. There are many strong arguments for a standard representation, but it has been hypothesized that if there were a simple way of expressing a limited set of transformations on data streams, that a large number of incompatibilities could be resolved and a great deal of computer-computer cooperation expedited.

The bulk of protocol work is being done with the invention of standard representations. The TELNET protocol, as discussed, is founded on the notion of a standard terminal called the Network Virtual Terminal (NVT). The File Transfer Protocol is working toward a standard sequential file (a Network Virtual File?). So it is also with less advanced protocol work in graphics and data management.

There is one experiment which is taking the transformational approach to dealing with incompatibilities. The Data Reconfiguration Service (DRS) is to be generally available for mediating between incompatible stream configurations as directed by user-supplied transformations.[13]

ACKNOWLEDGMENTS

Function-oriented protocols have been the principal concern of the ARPANET Network Working Group (NWG). A list of people who have contributed to the development of the protocols discussed would include, Robert Braden, Howard Brodie, Abhay Bhushan, Steve Carr, Vint Cerf, Will Crowther, Eric Harslem, Peggy Karp, Charles Kline, Douglas McKay, Alex McKenzie, John Melvin, Ed Meyer, Jim Michener, Tom O'Sullivan, Mike Padlipsky, Arie Shoshani, Bob Sundberg, Al Vezza, Dave Walden, Jim White, and Steve Wolfe. We would like to acknowledge the contribution of these researchers and others in the ARPA Network Working Group, without assigning any responsibility for the content of this paper.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
AFIPS Conference Proceedings May 1970

2 F E HEART et al
The interface message processor for the ARPA computer network
AFIPS Conference Proceedings May 1970

3 L KLEINROCK
Analytic and simulation methods in computer network design
AFIPS Conference Proceedings May 1970

4 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA Computer network
AFIPS Conference Proceedings May 1970

5 *Specifications for the interconnection of a Host and an IMP*
Bolt Beranek and Newman Inc Report No 1822
February 1971

6 C S CARR S D CROCKER V G CERF
HOST-HOST communication protocol in the ARPA Network
AFIPS Conference Proceedings May 1970

7 *HOST/HOST protocol for the ARPA Network*
ARPA Network Information Center #7147

8 T O'SULLIVAN et al
TELNET protocol
ARPA Network Working Group Request For Comments (RFC) #158 ARPA Network Information Center (NIC) #6768 May 1971

9 S M ORNSTEIN et al
The Terminal IMP for the ARPA Computer Network
AFIPS Conference Proceedings May 1972

10 J B POSTEL
Official initial connection protocol
ARPA Network Working Group Document #2 NIC #7101 June 1971

11 A K BHUSHAN et al
The data transfer protocol
RFC #264 NIC #7812 November 1971

12 A K BHUSHAN et al
The file transfer protocol
RFC #265 NIC #7813 November 1971

13 R ANDERSON et al
The data reconfiguration service—An experiment in adaptable process/process communication
The ACM/IEEE Second Symposium On Problems In The Optimization Of Data Communications Systems
October 1971

McROSS—A multi-computer programming system*

by ROBERT H. THOMAS and D. AUSTIN HENDERSON

Bolt, Beranek and Newman, Inc.
Cambridge, Massachusetts

INTRODUCTION

This paper describes an experimental "distributed" programming system which makes it possible to create multi-computer programs and to run them on computers connected by the ARPA computer network (ARPANET).[1] The programming system, which is called McROSS (for **M**ulti-**C**omputer **R**oute **O**riented **S**imulation **S**ystem), is an extension of a single-computer simulation system for modelling air traffic situations[2] developed by Bolt, Beranek and Newman, Inc. (BBN) as a tool for air traffic control research. The McROSS system provides two basic capabilities. One is the ability to program air traffic simulations composed of a number of "parts" which run in geographically separated computers, the distributed parts forming the nodes of a "simulator network." The second is the ability of such a simulator network to permit programs running at arbitrary sites in the ARPANET to "attach" to particular nodes in it for the purpose of remotely monitoring or controlling the node's operation.

The McROSS distributed programming system is unique in several ways:

(a) Use of McROSS generates inter-computer traffic in which a group of programs are engaged in substantive conversation. There is relatively little previous experience with such inter-computer, program-to-program conversations.

(b) The component nodes of a simulator network are not bound to particular ARPANET sites until simulation "run time." Thus on different runs the same distributed program can be distributed in different ways over the ARPANET. For example, in one run all the nodes of a simulator network might be run at BBN and on the next some might be run at BBN, others at RAND and still others at the University of Utah. This mode of using the ARPANET is significantly different from the normal one in which programs are bound to particular network sites at program composition time. (The only constraint on the binding of nodes to ARPANET sites is the requirement that each node run on a PDP-10 under the TENEX operating system.[5])

(c) The responsibilities of a node in a simulator network can be conveniently partitioned into subtasks which can be performed more or less independently of one another. The McROSS implementation mirrors this partitioning. Functions performed at the nodes are realized by groups of loosely connected, concurrently evolving processes.

The distributed simulation system represents an initial step in an on-going research program which is investigating techniques to make it easy to create, run and debug computations involving the coordinated behavior of many computers. The McROSS system is intended to serve both as an experimental vehicle for studying problems related to distributed computation and as a tool for air traffic control research. Its two goals are well matched. A satisfactory solution to the nation's air traffic problems is likely to include a network of airborne and ground based computers working together on a single distributed computation: the scheduling and control of aircraft maneuvers. Thus, the air traffic control problem is a rich source of interesting problems in partitioned computation which can be used to measure the usefulness of the distributed computational techniques developed.

This paper is a report on one phase of a continuing research program. The intent is to describe interesting aspects of an experimental distributed programming

* This work was supported by the Advanced Projects Research Agency of the Department of Defense under Contract No. DAHC-71-C-0088.

system and to share our experience with others considering the construction of such systems. The paper does no more than hint at how the McROSS system can be used in air traffic control research.

The next section provides background useful for subsequent discussion of the distributed programming system. After than, the McROSS system is described, first in terms of the facilities it provides for creating distributed simulations, and then in terms of interesting aspects of its implementation. The paper closes with a discussion of some of the issues brought into focus by the experience of implementing and using a distributed programming system.

COMPONENTS OF THE McROSS SYSTEM

The main components of the distributed programming system are:

1. The ARPA computer network;
2. The TENEX operating system for the PDP-10; and
3. A simulation programming system known as ROSS (for **R**oute **O**riented **S**imulation **S**ystem).

These components became operational within 6-10 months of one another, at which time it became feasible to implement the distributed programming system being described.

The ARPANET

The ARPA computer network[1,3,4] is a set of autonomous computer systems (*hosts*) which are interconnected to permit resource sharing between any pair of them. The goal of the ARPANET is for each host to make its resources accessible from other hosts in the net thereby permitting persons or programs residing at one network site to use data and programs that reside and run at any other. Each host interfaces with the network through an Interface Message Processor (*IMP*),[3] a small dedicated general-purpose computer. The IMPs, which reside at the various ARPANET sites, are connected by 50 kilobit common carrier lines and are programmed to implement a store and forward communication network. In passing from host A to host B a message passes from host A to IMP A, through the IMP communication network from IMP A to IMP B (in general passing through a number of intermediate IMPS) and finally from IMP B to host B.

At each host there is a Network Control Program (*NCP*) whose function is to provide an interface between the network and processes within the host. The NCPs use the IMP store and forward network to provide processes with a connection switching network. Thus, they enable processes in different hosts to establish connections with one another and to exchange information without directly concerning themselves with details of network implementation such as the way in which hosts are connected to and communicate with IMPs.

Information can flow in only one direction on an ARPANET *connection*. Thus, before two processes are able to engage in a dialogue they must establish two such connections between them. A connection is completely specified by its two ends which are called *sockets*. A network socket is itself uniquely specified by a host identifier and a socket identifier. The purpose of the socket identifier is to specify a process or group of processes within a host and a socket relative to that process or process group. Thus, it is useful to think of a socket as having a three component "socket name" of the form H.P.N. H is the "host" component which identifies an ARPANET host, P is the "process" component which identifies a process group within H and N is the "process-local" component which identifies a socket relative to P. In the sequel

$$H_1.P_1.N_1 \rightarrow H_2.P_2.N_2$$

is used to denote a connection between sockets $H_1.P_1.N_1$ and $H_2.P_2.N_2$ where → indicates the direction of information flow over the connection.

For a connection to be established between two processes each must request that it be established. There are two common ways in which connections are established. In the first, the processes play symmetric roles. Each specifies, as part of a "request for connection" (*RFC*), the socket name at its end of the connection, the socket name at the remote end of the connection and the direction of information flow. If the two RFCs "match" the connection is established. This connection procedure requires *a priori* agreement upon the sockets to be used for the connection. The second common connection procedure is used in situations in which one process wishes to provide some sort of service to other processes. The "serving" process establishes a *listening* socket within its host which is receptive to RFCs from any other process in the network and then "listens" for connection attempts. The serving process uses facilities provided by its NCP to detect the occurrence of a connection attempt and to determine its source. When such an attempt is made the serving process can choose to accept or reject it. This connection procedure requires that only one socket name, that of the serving process's listening socket, be known *a priori*. In the remainder of this paper

$$[\mathrm{H.P.N}\rightarrow]_{\mathrm{L}}$$

and

$$[\mathrm{H.P.N}\leftarrow]_{\mathrm{L}}$$

are used to denote connections established in a listening state.

The TENEX operating system

TENEX[5] is a time sharing system for the DEC PDP-10 processor augmented with paging hardware developed at BBN. For purposes of this paper it is useful to describe TENEX in terms of the virtual processor it implements for each logged-in user (i.e., user time sharing job).

The instruction repetoire of the TENEX virtual processor includes the PDP-10 instruction set with the exception of the direct I/O instructions. In addition, it includes instructions which provide access to virtual processor capabilities implemented by the combination of the TENEX software and hardware.

The TENEX virtual processor permits a user job to create a tree-structured hierarchy of processes. Such processes have independent memory spaces and computational power. At different stages in its lifetime a single user job may include different numbers of processes in various states of activity. Several mechanisms for interprocess communication are provided by the TENEX virtual machine. Processes can interact by sharing memory, by interrupts and by direct process control (e.g., one process stops, modifies and restarts another which is "inferior" to it in the hierarchy).

A memory space is provided by the virtual processor which is independent of the system configuration of core memory. Each process has a memory space of 256K words which is divided into 512 pages each of 512 words. A process can specify read, write and execute protection for pages in its memory space as it sees fit.

The virtual machine includes a file system which provides a mechanism for storing information on and retrieving it from external devices attached to TENEX. Processes refer to files using symbolic names, part of which identifies the particular device on which the file resides. The instruction set of the virtual machine includes operations for data transfer to and from files which a process can execute without explicitly arranging for buffering.

The NCP resident in TENEX makes the ARPANET appear to TENEX processes as an I/O device. The name of a "file" corresponding to a network connection includes the names of both the local socket and the remote socket which define the connection. A process requests TENEX to establish a connection by attempting to open an appropriately named "network" file. The open attempt succeeds and the connection is established if and when another process issues a matching RFC. TENEX processes transmit and receive information over network connections by executing the normal file data transfer instructions.

ROSS

ROSS is a programming system for modelling air traffic situations.[2] It includes facilities for creating and running simulation experiments involving air traffic in an experimenter-defined airspace. The system is currently being used in a number of air traffic control research projects.

To create a simulation experiment the experimenter uses ROSS language forms to write a "program" which defines the geography of an airspace, the wind profile for the airspace, aircraft flight characteristics and a set of "route procedures." A *route procedure* consists of a sequence of one or more "instructions" for monitoring or controlling the progress of an aircraft through the simulated airspace. Execution of a route procedure is the ROSS counterpart of pilot and/or controller actions. The "flight" of a simulated aircraft through the airspace is accomplished by the execution of a series of route procedures. ROSS includes "primitive" route procedures to "create" aircraft, "inject" them into and remove them from the simulated airspace as well as ones which cause aircraft to accelerate, turn, climb and descend.

By compiling his program the experimenter creates a simulator for traffic in the airspace he has defined. To perform a simulation experiment the experimenter runs his program. The simulated airspace remains empty until the program is supplied with input which inject aircraft into the airspace and control their flight through it. Each input line the simulator receives is passed to an internal parsing mechanism which issues a call to an appropriate experimenter-defined or primitive route procedure. The program can accept input from an on-line terminal, a predefined "scenario" file, or both. Input lines from the scenario file are identical to ones from the on-line keyboard with the exception that scenario file input lines include a time field which specifies the (simulated) time at which the program is to accept them. A user can manually "vector" aircraft through the airspace by supplying input at the on-line keyboard.

A ROSS simulator can drive displays of the airspace and, in addition, can generate output which can be written into files, typed on the on-line keyboard or sent back into the simulator input parser.

THE McROSS PROGRAMMING SYSTEM

The McROSS system provides the ability to define simulation experiments involving a number of airspaces or "simulation centers" which can be interconnected to form a simulator network. Adjacent centers in the simulator network are connected to one another by way of the ARPANET. The components of a simulator network may run as user jobs distributed among different TENEXs or as different user jobs on the same TENEX. (As of January 1972 the ARPANET included five TENEX hosts.)

Computational responsibility for performing a multicomputer McROSS simulation is truly distributed. For example, as an aircraft flies from one airspace into an adjacent one the responsibility for simulating its dynamics shifts from one computer to another.

Goals

The McROSS system was implemented to achieve the following goals:

1. Autonomy of parts:
   Individual components of a McROSS network should be able to operate independently of one another (to the extent that each is independent of the others for traffic). Furthermore, no center should be able to cause another to malfunction. Autonomy of parts enables a multi-computer simulation to run when only some of its components are operational. Failure of a component center in a multi-center simulation results in degradation of the total simulation rather than forced termination of it. A beneficial side effect of autonomy is that individual centers can be partially debugged without running the entire simulator network.
2. Deferral of process/processor binding:
   The binding of centers in a McROSS network to host computers in the ARPANET should be deferred until run time. This goal can be stated in more general terms. The program for a distributed computation defines a logical configuration made up of abstract relations between the computation's parts. A given execution of the program is accomplished by a particular physical configuration of computers. The two configurations, logical and physical, are by necessity related. However, the programmer should have the option of specifying them separately. By deferring process/processor binding the configurations can be separately specified. As a result the programmer is free while composing his program to concentrate on the logical structure he wants his program to define without concerning himself with details of the physical structure on which it is to be run.
3. Capability for dynamic reconfiguration:
   In the course of a simulation it should be possible for adjacent centers to dynamically break and reestablish connections with one another. Furthermore, it should be possible for process/processor binding to be changed during a simulation. That is, it should be possible to change the physical location of a center from one ARPANET host to another. The ability to dynamically reconfigure makes it possible to remove an improperly operating center from the simulator network and replace it with another at the same ARPANET host or at a different one.
4. Decentralization of control:
   McROSS is to be used as a tool for investigating distributed computation. Among the subjects to be studied are methods for controlling such computations. In particular, various techniques for distributing control responsibilities among the parts of a computation are to be experimentally investigated. It is important, therefore, that operation of the McROSS system not require a central control mechanism for coordinating simulator networks. Stated somewhat differently, the only components required for a McROSS simulation should be simulation centers defined by McROSS programs. The realization of this goal, which makes experimentation with distributed control possible, should not preclude experimentation with centralized control.
5. Remote monitoring capability:
   A McROSS simulator network should provide ports through which its components are accessible to any ARPANET host. An appropriately programmed process running at any ARPANET host should be able to "attach" to a component of a simulator network to monitor and control its operation. A remote monitoring process should be able to:

   a. obtain sufficient information to display traffic in the airspace it is monitoring.
   b. serve as the on-line keyboard for the center it is monitoring;

c. "detach" from one center and "attach" to another in the course of a simulation run.

McROSS as seen by the user

A McROSS simulator network is defined by a program composed of a "network geometry" sub-program and sub-programs corresponding to each of the centers in the network.

The network geometry sub-program defines the logical geometry for a simulator network. Conceptually, a network is composed of nodes and arcs. Nodes in a McROSS network are simulation centers and arcs are duplex connections between centers. Figure 1 shows a four node simulator network which could be used to simulate air traffic between Boston and New York. The following geometry sub-program defines that network:

```
netbegin
    netcen BOSTRM, BOSCEN, NYCEN,
           NYTRM
    netcon BOSTRM, BOSCEN
    netcon BOSCEN, NYCEN
    netcon NYCEN, NYTRM
netend
```

The *netcen* statement declares that the network contains four nodes (Boston terminal control, Boston en route control, New York en route control and New York terminal control). The *netcon* statements declare the three arcs; *netbegin* and *netend* serve to bracket the geometry declarations.

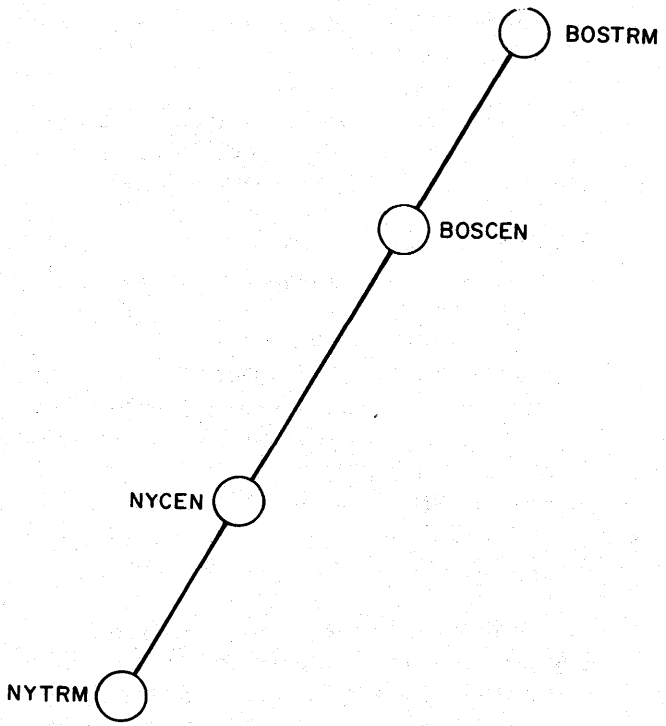


Figure 1—A simulator network which could be used to simulate air traffic between Boston and New York

In general, the sub-program for each center has four parts:

- a route procedure module
- a local geography module
- a wind profile module
- an aircraft characteristics module.

In addition to defining procedures followed by aircraft as they fly through a center's airspace, the route procedure module includes routines specifying how the center interacts with its neighbors. Information exchange between adjacent centers is accomplished by sending messages across the connection between them. A center handles messages from neighboring centers by submitting them to its input parsing mechanism. Such messages are treated identically to input from its on-line console and scenario file.

The ability of adjacent centers to interact depends upon the state of the connection between them as each sees it from its end of the connection. A center may consider a connection to be in one of three states:

1. uninitialized:
   the "physical location" of the neighbor is unknown;
2. closed:
   the "physical location" of the neighbor is known but the connection is not capable of carrying messages (either because it has not yet been established or because it has been broken);
3. open:
   the connection may be used for information exchange with the neighbor.

In the current implementation of McROSS the "physical location" of a neighbor includes both the ARPANET host the center is running on (e.g., BBN) and the identification of the user it is running under (e.g., Jones).

McROSS provides the operations *init*, *conn*, *dsconn* and *abort* for changing the state of a connection. The effect these operations have on the end of a connection is illustrated by the state transition diagram of Figure 2.

Consider the geometry for the Boston-New York simulation. Execution of

conn BOSTRM

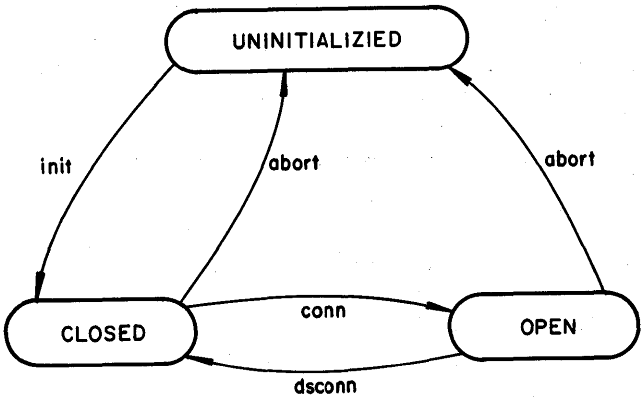


Figure 2—Transition diagram for the state of the end of a connection showing the effect of the operations *init, conn, dsconn* and *abort*

within the BOSCEN initiates an attempt to open a connection with BOSTRM. The connection attempt succeeds if a matching *conn* is executed by the BOSTRM center. The effect of executing

dsconn NYCEN

within the BOSCEN center simulator is to break the connection between the NYCEN and BOSCEN centers by forcing both ends of it into the closed state; *abort* works in an analogous manner. Execution of the *init* operation results in a center-user dialogue in which the human user is asked by the center program to specify the physical location of the neighbor.

Two language primitives are provided for sending messages from one center to another. One takes a single operand, a message, which it sends to every neighbor whose connection is in the open state. The other takes two operands, a message and the name of a neighboring center. If the connection to the center is open, the message is sent to the center; otherwise the operation has no effect. Because they are submitted to the input parsing mechanism care must be taken that messages sent to a neighbor are formatted correctly.

McROSS includes operations which can be used by a center to obtain information about the simulator network and its immediate neighbors. For example, there is a primitive which produces a list of all nodes in the network (i.e., all centers declared by the *netcen* declaration); another one produces a list of all neighboring centers for which connections are in the open state. In addition, a center can examine the state of the connection to each of its neighbors individually.

The local geography module defines the airspace of a center by specifying names and locations (x-y coordinates) of important geographic features such as navigational aids, obstructions and airports. In addition it includes a declarative statement which names the simulation center. For example, the geography module for the BOSTRM center would include the declaration

atccen BOSTRM.

This declaration has the effect of binding the identifier THISIS to the name BOSTRM. Thus in the BOSTRM center THISIS is bound to BOSTRM while in the NYTRM center it is bound to NYTRM. Route procedures which access the simulator center name can be written in terms of THISIS to enable their use at any center in a simulator network.

A properly authorized process at any ARPANET host can attach to a center in a McROSS simulator network and request output from it and direct input to it thereby monitoring and controlling its operation. McROSS center programs are prepared to offer two kinds of service to remote monitors:

1. broadcast service:
   Centers continuously broadcast certain information to monitors attached to them. Presently centers broadcast flight parameters of each aircraft in their air space (speed, heading, altitude, x-position, y-position, acceleration, aircraft id) and the (simulated) time. A remote monitor can use broadcast information to drive a display of traffic in a center's airspace or it can record it for later analysis.
2. demand service:
   Each center is prepared to respond to certain specific requests from monitors. In the current implementation a monitor can request that a center:
   a. transmit its map of the airspace (which can be used as background for displaying the center's air traffic);
   b. stop the continuous broadcast;
   c. resume the continuous broadcast;
   d. treat the monitor as its on-line keyboard by directing keyboard output to the monitor and accepting input from it;
   e. cease treating the monitor as its on-line keyboard;
   f. break its connection with the monitor.

The monitoring facility has proven useful both for debugging and for demonstration purposes. One difficulty a user faces in debugging a multi-center simulation is determining what is happening at centers suspected to be malfunctioning. A monitor, constructed appropriately, can serve as a "graphical probe" and be used to watch the operation of first one suspect center and then another. For example, we have used such a monitor to follow the trajectory of an aircraft as it passes through several centers.

By enabling processes at arbitrary ARPANET sites to observe and control McROSS simulations, the monitoring facility provides a mechanism for using hardware and software features which are unique to various network installations. By using monitors which play an active role in his simulations a McROSS user can experiment with different ways of partitioning computational and control responsibilities in air traffic situations. He could, for example, experiment with a monitor built to provide a weather advisory service for simulator centers. Such a monitor would presumably have access to an on-line weather data base. (A weather data base to be accessible through the ARPANET is currently being designed.[10]) To perform its service for a center the monitor would attach to the center, requesting that the center broadcast aircraft flight parameters to it and accept input lines from it. It would then "watch" the airspace of the center and send instructions to it, as necessary, to vector aircraft around severe weather.

Unless he chooses to do so, the simulation programmer need not concern himself with remote monitoring beyond specifying at simulation run time which centers in his network are to be receptive to remote monitors. Monitors themselves are not part of the McROSS system. McROSS merely provides a mechanism for remote monitoring. No effort has been made to provide linguistic features within the McROSS system to make it easy to write programs to do monitoring.

THE McROSS IMPLEMENTATION

Some interesting aspects of the McROSS implementation are discussed in this section. The section focuses on strategy rather than detail. The result is a simplified but nonetheless accurate sketch of the implementation.

The ROSS implementation

Implementation of McROSS was accomplished by extending the existing ROSS simulation system. A ROSS simulation consists of initialization, simulation and termination phases. The simulation phase is implemented as a loop. Each pass through the loop represents a "tick" of the clock which maintains simulation time. On each tick the simulator:

1. parses and interprets input directed to it from its scenario file and on-line console since the last tick;
2. interprets the route procedure each aircraft is currently following;
3. advances each aircraft along its trajectory taking into account the aircraft's speed, acceleration and heading and the wind profile of the airspace;
4. directs output generated since the last tick to the appropriate devices;
5. performs actions necessary to maintain the local display of its airspace:
6. increments the simulated time.

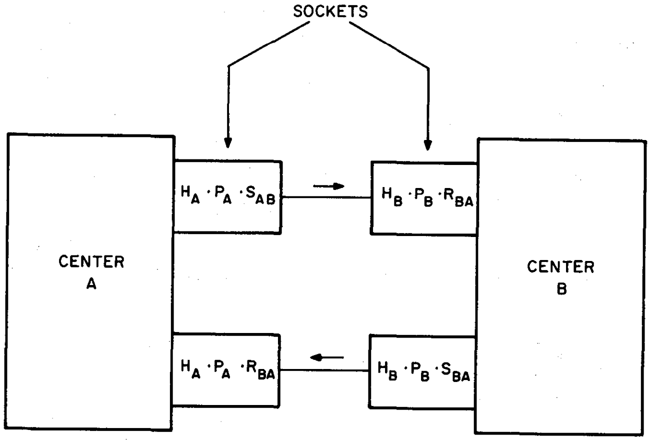

SOCKETS
CENTER A
$H_A \cdot P_A \cdot S_{AB}$
$H_B \cdot P_B \cdot R_{BA}$
CENTER B
$H_A \cdot P_A \cdot R_{BA}$
$H_B \cdot P_B \cdot S_{BA}$

Figure 3—Schematic of center-center connection between adjacent centers A and B. S_{AB}, R_{AB}, S_{BA} and R_{BA} are assigned at program translation time; H_A, H_B, P_A and P_B are determined at run time

A ROSS simulation can be run either in a "real time" mode in which simulated time is locked to real time, or in a "hyper fast" mode in which the simulation proceeds as fast as it can.

Process/processor binding and center-center connections

Connections between pairs of simulator network centers are duplex. Each center-center connection is implemented by two ARPANET connections. More specifically, an open connection between centers A and B is realized by the two ARPANET connections (see Figure 3):

$$H_A.P_A.S_{AB} \rightarrow H_B.P_B.R_{BA}$$

$$H_A.P_A.R_{AB} \leftarrow H_B.P_B.S_{BA}$$

To establish such a center-center connection two pairs of matching RFCs must be issued by centers A and B. To issue the correct RFCs A must know H_B, P_B, R_{BA} and S_{BA}; similarly, B must know H_A, P_A, R_{AB} and S_{AB}.

The host (H) and process (P) components of the socket names for a center-center connection cannot be determined until run-time because process/processor binding is deferred until then. However, the process-local components (R and S) of the socket names can be pre-assigned and, in fact, the effect of the declarations in the network geometry sub-program for a particular McROSS network is to do exactly that.

The process local components for the four socket names corresponding to a center-center connection are always the same whereas the host and process components may change from run to run or even within the same run if either neighbor is involved in a dynamic reconfiguration.

When a center's end of a center-center connection is in the uninitialized state the host and process components of the socket names corresponding to the remote end are unknown to it. To move its end of a connection from the uninitialized state the center engages in a dialogue with the user requesting from him the physical location of the neighbor. After successfully completing the dialogue the center has sufficient information to issue the two RFCs required of it to establish the connection.

Center-monitor connections

The connection between a center C and an attached remote monitor M is realized by two ARPANET connections. One of them

$$H_C.P_C.S_{CM}\rightarrow H_M.P_M.R_{MC}$$

is a "broadcast" connection used for continuously broadcasting information to M. The other

$$[H_C.P_C.R\leftarrow]_L$$

is a "request" connection maintained by C in a listening state for M to use to make requests of C. Each monitor attached to C has its own broadcast connection but may share a request connection with other monitors.

To make a request of C, M connects to the request socket, transmits its request over it and then closes the request connection, freeing it for use by other monitors. The action taken by C of course depends upon the request. If the request were for the display map, C would transmit the map data over the broadcast connection to M.

Before the RFCs required to establish a center-monitor connection between C and M can be issued C must know H_M, P_M and R_{MC} and M must know H_C, P_C, S_{CM} and R. To obtain the required information, M and C engage in a connection protocol which is similar to the "initial connection protocol" developed by the ARPANET network working group.[6]

Each center willing to service monitors maintains as an "attach" socket a send socket in a listening state (see Figure 4.a). The attach socket for C could be denoted

$$[H_C.P_C.A\rightarrow]_L$$

The process local component (A) of the name for attach sockets is the same for all centers and is "well-advertised." Therefore, if M knows the physical location of C it can issue an RFC for C's attach socket. The effect of such an RFC is to establish the connection (Figure 4.b)

$$H_C.P_C.A\rightarrow H_M.P_M.R_M$$

Upon detecting an RFC for its attach socket, C notes H_M and P_M and transmits S_{CM} and R over the connection. Next both C and M break the connection and

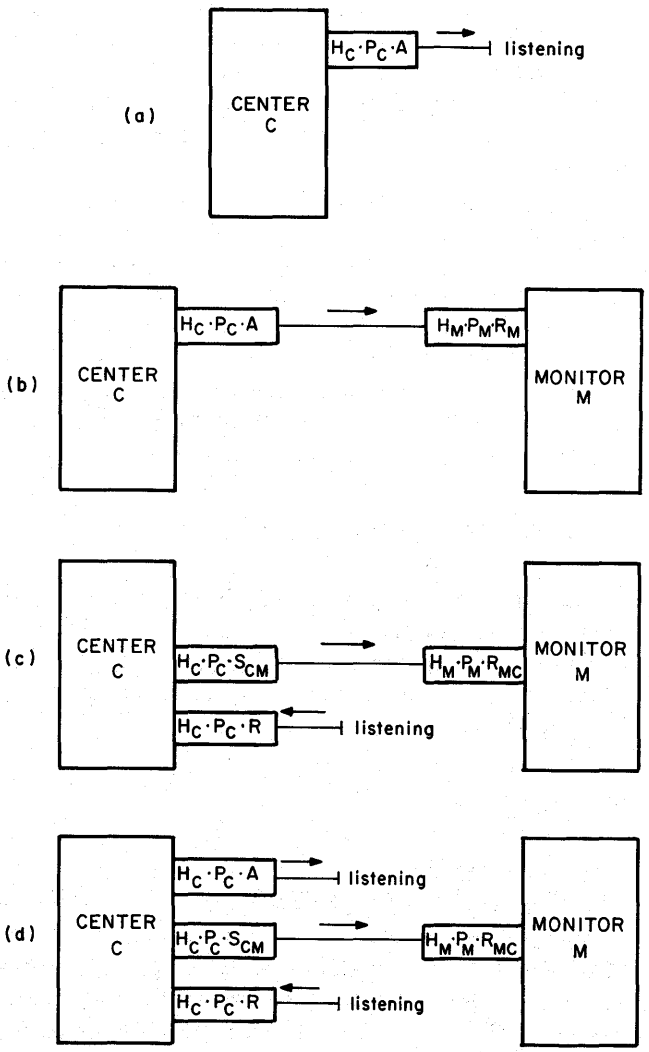


Figure 4—Schematic of connection protocol exchange between center C and monitor M

issue the RFCs necessary to establish the broadcast connection (Figure 4.c)

$$H_C.P_C.S_{CM} \rightarrow H_M.P_M.R_{MC}$$

where $R_{MC}=f(R_M)$, a pre-agreed upon function of R_M. C, if it has not done so previously for another monitor, sets up the listening connection

$$[H_C.P_C.R \leftarrow]_L$$

and finally, reestablishes its attach connection so that other monitors can attach to it (Figure 4.d). Currently the process-local components for ARPANET socket names are numbers and f is taken to be

$$f(x)=x+2.$$

If R_M rather than $f(R_M)$ were used for R_{MC} a race condition would result when M and C attempt to establish the broadcast connection. The race involves the order in which the connection with socket $H_M.P_M.R_M$ would be closed and reopened by C and M. In particular, the current TENEX NCP is such that an attempt by C to open the network file corresponding to its end of the (broadcast) connection

$$H_C.P_C.S_{CM} \rightarrow H_M.P_M.R_M$$

before M closes the network file corresponding to its end of the connection

$$H_C.P_C.A \rightarrow H_M.P_M.R_M$$

would fail. Use of $f(R_M)$ for R_{MC} avoids the race. A similar race condition, discovered in an early version of the ARPANET initial connection protocol,[7] is avoided in the current protocol by the same technique.[8]

Process structure at McROSS centers

A McROSS center is realized by a collection of cooperating, asynchronously evolving, sequential processes. The collection corresponds to a partitioning of the center's responsibilities into more or less independent sub-tasks. It includes:

1. a process (SIM) to perform the ROSS functions;
2. a process (CONN) for each center-center connection to establish and maintain the connection; and
3. a monitor server process (MONSER) to service remote monitors.

The CONN process at center A corresponding to the center-center connection to center B is responsible for establishing ARPANET send and receive connections with B. After it establishes the center-center connection with B the CONN process maintains the connection. When messages from B arrive it passes them on to the SIM process for parsing.

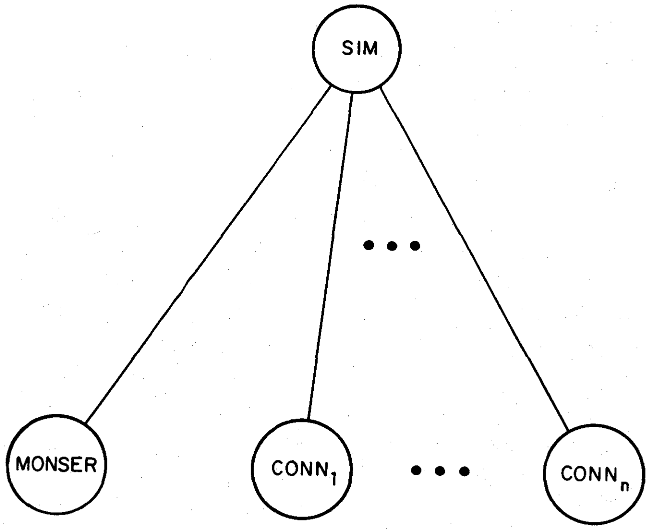


Figure 5—The process hierarchy which implements a McROSS simulation center with *n* neighbors

The job of the MONSER process at a center is twofold: to engage in an initial connection protocol exchange with monitors attempting to attach to the center and to respond to requests made by attached monitors.

The processes at a center exist in a hierarchy (see Figure 5). The hierarchical structure is less the result of any one process being more important than any other than it is a consequence of the TENEX constraint that process groups be hierarchically arranged. During initialization the SIM process creates the MONSER and CONN processes. Thereafter, the processes evolve more or less independently.

The process structure at each center helps achieve autonomy of parts. The CONN processes and the MONSER process serve to protect the SIM process by insulating it from direct interaction with other centers and remote monitors.

Protocols

The current implementation of the TENEX NCP is such that if a center were to unilaterally break a connection with a neighbor (by closing the two corresponding ARPANET connections) it could leave processes in the neighboring center in an irrecoverable state. For example, a process in the neighbor sending information across the connection at the time it is broken would "feel" the close as if it had executed an illegal instruction. To prevent such situations McROSS centers engage in a protocol exchange prior to breaking connections.

The center-center protocol is relatively simple. To perform an *abort* or *dsconn* operation a center sends its neighbor a "request for abort" or "request for disconnect" message and waits until it receives an acknowledgment before actually breaking the connection. Existence of the center-center protocol has two major implications. The first is that a center-center connection has more states than the three noted earlier. The additional states are transient ones which the connection passes through as the center and its neighbor advance through protocol exchanges initiated when one attempts to change the state of the connection. The transient states are invisible to the McROSS user. Immediately after a *dsconn* (or *abort*) is initiated the SIM process treats subsequent operations involving the connection as if the connection were already in the closed (or uninitialized) state. The second implication is that center-center connections carry "control" messages used in center-center protocol exchange in addition to "ordinary" messages intended for the receiver's parsing mechanism. The CONN process for each connection must be prepared to recognize and respond appropriately to both kinds of messages.

McROSS centers expect remote monitors to observe a center-monitor protocol. In addition to the connection and request procedures described earlier, the center-monitor protocol includes a disconnection procedure much like the one used in the center-center protocol.

Interprocess communication

To perform their tasks the processes at a simulation center must interact occasionally. For example, the arrival of a message from a neighbor requires interaction between the SIM and CONN processes. The CONN process receives the message and passes it onto the SIM process for parsing and interpretation.

One way the processes interact is through shared memory. For example, the SIM and CONN processes have read and write access to a shared "connection table." There is an entry in the table for each center-center connection which includes the state of the connection, a semaphore[9] and other information relevent to the connection. Use of the table is coordinated by strict adherence to a convention which requires every "critical" access (every write access and certain read accesses) to an entry to be bracketed by P (lock) and V (unlock) operations on the entry's semaphore.

The situation arising at a center when a neighbor attempts to break connection with the center is a typical one which requires interprocess communication. The CONN process corresponding to the center-center connection receives a "request for disconnect" message from the neighbor. Center-center protocol requires the CONN process acknowledge the request so that the neighbor can break the connection. The purpose of the protocol exchange is to warn the center that the connection is about to be broken and that it should no longer attempt to use it. Therefore before it acknowledges the neighbor's request it is important that the CONN process communicate this information to the other processes at its center. The CONN process does this by the following sequence of actions:

```
P(CONNECTION SEMAPHORE);
set connection-state to "not open";
V(CONNECTION SEMAPHORE);
acknowledge neighbor's request
```

As long as processes at the center perform the following sequence of actions to send over center-center connections there is no danger of sending after the connection has been closed:

```
P(CONNECTION SEMAPHORE);
if connection-state = open
    then send
    else abort the send;
V(CONNECTION SEMAPHORE)
```

Stated somewhat differently, sending over a center-center connection should be regarded as an operation which involves a "critical" read access of the corresponding connection table entry.

In addition to memory sharing, direct process control is used as a mechanism for interprocess communication in the McROSS system. Because of its superior position in the process hierarchy the SIM process can exert direct control over the other processes. A few situations occur in which it does so. For example, when a center-center connection has been closed or aborted (via *dsconn* or *abort*) the SIM process forces the corresponding CONN process to halt. If and when an attempt is initiated to reestablish the connection (via *conn*) SIM restarts it.

SOME OPEN QUESTIONS

This section briefly discusses questions representative of ones which have arisen in the course of using the McROSS system. The questions have been resolved to the extent that useful simulations can be performed using McROSS. However, none has been resolved in a totally satisfactory manner. The intent of this section is to leave the reader with an appreciation for the issues raised by these questions; a thorough discussion of them is well beyond the scope of this paper.

Synchronization

Simulated time is important in the operation of the McROSS system. In particular, whenever an interaction between adjacent centers occurs it is important that the clocks kept by the centers show approximately the same time. Time synchronization is a specific example of the general problem of control in distributed computation. It is compounded by the fact that centers can start up and shut down independently of one another. A centralized approach to synchronization has been used with success in McROSS simulations. In it, one center acts as a synchronizer for an entire simulator network. When a center starts up it connects to the synchronizer and receives a synchronization message from it. Thereafter, to stay in synch with other centers in the network, the center makes use of the real time clock in the computer it runs on. A distributed approach to synchronization which does not require a synchronizing center is under consideration.

Locally unknown names

Names that are well defined within a simulator network as a whole are not necessarily defined at every node in the network. How should references to such names occurring within centers in which they are not defined be handled? For a specific example in which such a reference is reasonable, reconsider the four node network for simulating Boston-New York traffic. A user controlling the simulator for Boston Terminal who is manually vectoring an aircraft leaving Logan airport might reasonably issue the clearance

fly (V205, PAWLING)

which specifies that the aircraft is to follow Victor Airway #205 to Pawling. Assume that V205 is defined within the geography modules for BOSTRM, BOSCEN and NYCEN and the PAWLING is defined within NYCEN but not within BOSTRM or BOSCEN. The BOSTRM center can't fly the aircraft to Pawling because Pawling is not defined within its airspace. Ideally it should fly the aircraft along V205 to the boundary of the BOSCEN airspace and then hand it off to the BOSCEN simulator. Certainly it should be able to do better than report an error and abort the route procedure. Techniques for handling references to locally unknown names in certain limited contexts are being investigated. However, the general problem of handling such references is an open question.

Program residence

Where should the program (route procedures) required to fly an aircraft through several simulator centers reside? Should the program be associated with the aircraft and passed with it from center to center or should parts of the program be distributed among the relevant centers? The approach currently used in McROSS simulations is to distribute the program and pass only the aircraft and its flight parameters from center to center.

Interruption and resumption of route procedures

Aircraft frequently interrupt their flight plans temporarily in order to avoid severe weather or heavy traffic. The simulation analogy to a flight plan is the route procedure. How should a center handle interruption and subsequent resumption of route procedures? Interrupting the execution of a route procedure in order to follow another one is not difficult. The difficulty arises in determining how to appropriately resume the interrupted procedure. In general, the point of interruption is inappropriate as a resumption point. A considerable amount of (simulated) time can elapse between interruption and resumption during which the flight parameters (position, speed, altitude and heading) of the aircraft can change significantly. Therefore, the usual programming technique of saving the "state" when an interrupt occurs and restoring it after the interrupt has been handled is inadequate. The interruption/resumption problem is made more complex by the possibility that between interruption and resumption the aircraft may fly out of the airspace of one center and into the airspace of another. The current McROSS implementation is not prepared to handle interruption and subsequent resumption of route procedures.

Error handling techniques for distributed systems

The question of how to handle error situations in a distributed computational system is a challenging one. In McROSS considerable attention has been given to making nodes in a simulator network autonomous. The strategy for handling errors is to try to achieve a "local" error recovery whereby a node attempts to preserve its autonomy. As a result, while the actions it takes are locally optimal in the sense that its continued operation is insured, they may be sub-optimal in the more global context of the entire simulation network.

Errors occurring in inter-node messages are simply handled in the current McROSS implementation. Recall from an earlier section that inter-node messages are submitted to the parsing mechanism of the destination node. When a node receives a message which is syntactically incorrect or semantically meaningless (to it) from a neighbor, it reports the error on its on-line keyboard, sends a message to the neighbor causing the error to be reported on the neighbor's on-line keyboard, and ignores the message. This procedure is locally satisfactory in the sense that it guarantees that messages which are not well-formed cannot cause the node to malfunction. However, if the incorrect message from the neighbor is one critical to the outcome of the simulation, this procedure is not globally acceptable. Ideally, upon detecting an error in a message, the node should engage in a dialogue with its neighbor in an attempt to resolve the error. The difficulty in implementing this strategy is that it is frequently unclear what should be done to resolve an error. Often the cause has been masked by the time the error is detected.

While the simple techniques used in McROSS for error handling have proven adequate, it is clear that more effective techniques can be developed.

CONCLUDING REMARKS

The results of the work reported in this paper are applicable in two areas.

One area is research in air traffic control. Researchers can use the McROSS system to conduct simulation studies preliminary to the design of an automated multi-component air traffic control system. For example, McROSS could be used to evaluate various ways of partitioning responsibility among components of such a system. Or, it could be used to compare different strategies for automated scheduling and control of aircraft. Because it exhibits autonomy of parts and the ability to dynamically reconfigure, it could be used to experimentally study the behavior of failure handling techniques for a multi-center system under various air traffic loads.

The other area is the design and implementation of distributed computation systems. The results in this area are incomplete and tentative consisting primarily of insights and attitudes developed from the experience of building and using a distributed computation system. These are summarized by reviewing the goals for the McROSS system in terms of problems they posed and techniques useful in realizing them.

Of the five goals, autonomy of parts and deferral of process/processor binding were the most significant in terms of effort required to achieve them and influence on the appearance of the system to users. Given their realization, the other three goals (the ability to dynamically reconfigure a simulator network, decentralization of control and ports for monitoring) were relatively easy to achieve.

The strategy of implementing parts of the distributed computation by process groups rather than solitary processes contributed significantly to achieving autonomy of parts. The multi-process implementation made it possible to dedicate certain processes at a node to maintaining an interface with other nodes and to dedicate other processes to performing functions crucial to the node's existence. In addition to insulating vital internal node functions from the actions of other nodes, the functional modularity resulting from multi-process nodes had the effect of reducing the complexity of the implementation: each individual process being considerably less complex than an entire node. The multi-process capability which the TENEX operating system supports for each user job was invaluable in carrying out the multi-process strategy. It is unfortunate that few operating systems allow users to create and maintain process structures.

Equally useful in realizing autonomy was the establishment of and strict adherence to protocols for part-part interactions. A center can expect monitors and adjacent centers which are functioning properly to observe protocol and can therefore interpret a breach of protocol as a warning that the offender may be malfunctioning. A consequence of the multi-process implementation of nodes is that interprocess communication occurs within McROSS at two levels: inter-node and intra-node. Use of a protocol for intra-node interactions helps insure that internal node data bases always accurately reflect the condition of the node's interface with other nodes. A useful implementation rule was to reject any technique whose success depended upon the order in which events in different centers or in different processes within the same center occur.

The major problem in implementing deferred process/processor binding was providing a way for parts of the computation to determine the location of logically adjacent parts at run time. The solution used in the current McROSS implementation, which requires run time interaction with the user, is not totally satisfactory. A more satisfactory alternative might be for each part to engage in a network-wide search for logically adjacent parts.

We expect to see a trend toward distributed multi-computer systems in the future. By its existence McROSS demonstrates that the construction of such systems is currently feasible.

ACKNOWLEDGMENT

The McROSS system grew from an idea proposed by Dr. W. R. Sutherland of BBN; we gratefully acknowledge his guidance and enthusiasm.

The realization of the current McROSS implementation is the result of building upon the work of others, most significantly the designers and implementers of the ARPA computer network and of the TENEX operating system.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
Proceedings of AFIPS SJCC 1970

2 W R SUTHERLAND T H MYER E L THOMAS D A HENDERSON
A route oriented simulation system for air traffic control studies
Proceedings of the Fifth Conference on Applications of Simulation December 8-10 1971

3 F E HEART R E KAHN S M ORNSTEIN W R CROWTHER D C WALDEN
The interface message processor for the ARPA network
Proceeding of AFIPS SJCC 1970

4 S CARR S CROCKER V CERF
HOST-HOST protocol in the ARPA computer network
Proceedings of AFIPS SJCC 1970

5 D G BOBROW J D BURCHFIEL D L MURPHY R S TOMLINSON
TENEX, A paged time sharing system for the PDP-10
Paper presented at the Third ACM Symposium on Operating System Principles October 18-20 1971 published in Communications of the ACM March 1972

6 *ARPA network current network protocols*
August 1971 Available from the Network Information Center as NIC #7104 at Stanford Research Institute Menlo Park California 94025

7 W NAYLOR J WONG C KLINE J POSTEL
Regarding proffered official ICP
Unpublished memo available from Network Information Center NIC #6728, 1971

8 A SHOSHANI
A solution to the race condition in the ICP
Unpublished memo available from the Network Information Center NIC #6772 1971

9 E W DIJKSTRA
Cooperating sequential processes
Appears in Programming Languages edited by F Genuys Academic Press New York 1968. Also published as Report EWD123 Dept of Mathematics Technological University Eindhoven The Netherlands 1965

10 D STERN
Weather data
Unpublished memo available from Network Information Center NIC #7692 1971

Extensions of packet communication technology to a hand held personal terminal

by LAWRENCE G. ROBERTS

Advanced Research Projects Agency
Arlington, Virginia

INTRODUCTION

Electronic communications technology has developed historically almost completely within what might be called the circuit switching domain. Not until the last decade has the other basic mode of communication, packet switching, become competitive. Thus, as a technology, packet communication has only begun to be explored. Circuit switching can be defined in the broad sense as the technique of establishing a complete path between two parties for as long as they wish to communicate, whereas packet switching is where the communication is broken up into small messages or packets, attaching to each packet of information its source and destination and sending each of these packets off independently and asynchronously to find its way to the destination. In circuit switching all conflicts and allocations of resources must be made before the circuit can be established thereby permitting the traffic to flow with no conflicts. In packet switching there is no dedication of resources and conflict resolution occurs during the actual flow perhaps resulting in somewhat uneven delays being encountered by the traffic. Clearly, without the speed and capability of modern computers, circuit switching represented a cheaper and more effective way to handle communications. For radio frequency assignment and telephone exchanges the resource allocation decisions could be made infrequently enough that manual techniques were originally sufficient. Also, since voice was the main information being communicated, the traffic statistics were sufficiently compatible with this approach to make it quite economic for the period. Packet switching of a kind, the telegram, persisted throughout this period but due to the high cost of switching and the limited demand for fast message traffic never attracted much attention.

For almost a century circuit switching dominated the communications field and thus dominated the development of communications theory and technology. Now, within the last decade or less, the advances in digital computers and electronics have, in many cases, reversed the economic balance between circuit and packet communication technology. Perhaps the best proof of this is the economy of the ARPA Network[1–6] for country-wide computer to computer communication, but many other examples are beginning to appear such as the University of Hawaii's ALOHA System[7] utilizing packet radio transmission for console communications and the experiments with digital loops for local distribution. However, most of the experiments with packet communications have been undertaken by computer scientists, and it is not even generally recognized yet in the communications field that a revolution is taking place. Even where the knowledge of one of these experiments has penetrated the communications field, it is generally written off as a possibly useful new twist in communications utilization, and not recognized as a very different technology requiring a whole new body of theory. Throughout the development of the ARPA Network, communication engineers compared it with conventional circuit switched systems but, perhaps unconsciously, used rules of thumb, statistics and experience applicable only to circuit switched systems as a basis for comparison. A century of experience and tradition is not easy to ignore and in fact should not be ignored, only it should be properly classified and segregated as resulting from a different technology.

Packet communication technology is only beginning to be explored but already it is clear that the design of all forms of communications channels and systems should be rethought. As an example of the kind of difference packet communications can make in a perhaps unexpected area, the design of a personal terminal will be explored in some detail. Although such a terminal has never been built, it is most likely completely feasible to build and would provide many unique advantages.

HAND HELD PERSONAL TERMINAL

Let us start with the goal of providing each individual with a pocket-sized, highly reliable and secure communications device which would permit him to send and receive messages to other individuals or to coputers. Leaving the consideration of design alternatives muntil the end, a device fulfilling these objectives is as follows:

Output

Text or graphics displayed on a 2.8″×1″ plasma panel with 80 dots per inch resolution. The screen, divided into 7×10 dot rectangles, using 5×7 characters would hold 8 lines of 32 characters each for a total of 256 characters. Text this size is almost the same size as typewriter print, except that the lines are closer together. The plasma panel has internal storage and is digitally addressed to write or erase a point.

Input

Five capacity or stress sensitive buttons used by the five fingers of one hand simultaneously to indicate one of 31 characters. This five finger keyboard technique was developed by Doug Englebart at SRI[8] to permit users to type with only one hand while working on a display console. Recently the keyboard has become fairly widely used at SRI due to its great convenience. Training time for a new user is evidently less than a day and speeds of 30 words per minute can be achieved.[9] Although somewhat slower than a good typist (½ speed) the speed is clearly sufficient for a terminal device even at 10 words/minute.

Transmission

Each input character will be transmitted to a central controller station using the random access radio transmission techniques developed at the University of Hawaii.[7] The 5 bit character is embodied in a 64 bit packet containing:

30 bits—Terminal Identification Number
8 bits—Character plus alternation bit, or Count
2 bits—Type of packet (CHAR, ACK, CNT, ERR, ST ERR)
24 bits—Cyclic Sum Check
64 bits

All terminals transmit their packets independently and asynchronously on a single frequency and the receiver at the central controller merely listens for a complete packet which has a correct sum check. If two terminals' transmissions overlap the sum check will be wrong, and the terminals will retransmit when they find they don't receive an acknowledgment. Retransmission time-out intervals are randomized between the terminals to avoid recurrence of the problem. Upon receipt of a good packet, the central station transmits a display-acknowledgment packet back to the terminal on a second frequency. This 144 bit packet contains a 70 bit display raster field and an 8 bit position on the screen. The display raster is a 7×10 dot array for the character sent in and the position includes 3 bits for vertical by 5 bits for horizontal. Current position information for each active user is kept by the central station by user ID in a hash table. Thus, the individual terminal needs no character generation logic, position advancement logic, or any other local verification of the input since the response from the central station both acknowledges the character and displays it in an input text line at the top of the display. If a character display-acknowledgment is somehow lost in transmission the terminal will continue to time-out and retransmit the character. The central station must somehow differentiate this from a new character. This is achieved by an alternation bit[10,11] in the terminal's packet which is complemented for each *new* character. On a repeat the bit is the same as previously and the central station just retransmits the same character and position again. When a prearranged terminating character is sent the central station examines the message and takes an appropriate action. Considerable flexibility exists here, and operational modes could be established. However, the first message of a sequence should contain a destination as the first item. This might be the ID of another terminal in the same area, it might be the address of a service computer or it might be the ID of another terminal halfway around the world. In the latter two cases, a more global network such as the ARPA Network comes into play. It would be perfectly feasible for a message to another terminal to be sent to a central or area-coded directory computer to locate the particular control station the other terminal was near. Note that the location of neither man was given to the other, only the message and the ID of the sender. (Based on ARPA Network cost estimates and international satellite tariff trends, such a message exchange should cost less than 0.1 cents, independent of distance.)

Reception

At any time when a message destined for a terminal arrives at the central control station, a transmission to the terminal may begin, character by character, each in its own 144 bit packet as follows:

30 bits—Terminal Identification Number
70 bits—7×10 dot pattern, character display
8 bits—position of character
1 bit —alternation bit
1 bit —broadcast mode
3 bits—Message Type (Response, initial, normal)
8 bits—Characters Left in message
24 bits—Cyclic Sum Check
144 bits

The terminal must always be checking all transmission to detect those with its ID and a correct sum check. When one occurs which is not a "response" to an input character, a message is being sent. The first character of a message is marked type "initial," and has the count of the remaining characters. Each character is displayed where the central station placed it. Following the "initial" character "normal" characters are checked to make sure the count only decreases by one each time. After the character with count zero, an acknowledgment type packet is sent by the terminal. If this is lost (as it may be due to conflicts) the central control will retransmit the final character over again without complementing the alternation bit until it is acknowledged (or it determines the station is dead). If a count is skipped the terminal sends a CNT ERR message with the count of the character expected. The transmitter then starts over at that count. If a "normal" type character is received before an "initial" type a ST ERR message is sent and the message is restarted. A broadcase bit is included which overrides the ID check for general messages.

Security

Since all transmissions are digital, encryption is possible and would be important no matter what the application, military or civilian. Most private uses such as personal name files, income-expense records, family conversations, etc., would be far more sensitive than current computer console use.

Bandwidth

Personal terminals for occasional use for message exchange, maintaining personal files, querying computer data bases for reference data, etc., would not lead to very heavy use, probably no more than two query-responses per hour. The query we might estimate at 64 characters in length and the response at 256. (Clearly 256 character response could also consist of an 80×224 point graphic display since each character is sent as a full 7×10 raster.) The average bandwidth consumed by each terminal is therefore 2.3 bits/second transmitted and 25.6 bits/second received. The random access technique used for transmission requires the channel bandwidth to be six times the average bandwidth actually utilized in order to resolve all conflicts properly. Thus, the terminal transmission bandwidth consumption is 14 bits/second, still less than the receiver bandwidth needed. Thus, the central control station's transmitter bandwidth is the limiting factor assuming equal bandwidths on both transmitter and receiver. If a 50KHz bandwidth is used for each and modulated at 50K bits/sec, then a total of 2000 terminals can be accommodated. Of course this number depends on the activity factor. At one interaction every two minutes a data rate equal to average time shared console use is obtained and even at this activity 130 terminals can be supported, more than most time-sharing systems can support. With 50 KB channels, the time required to write 256 characters is about one second. Lower bandwidths require increased time, thus, 10KB (5 sec write time) would be the lowest bandwidth reasonable. Even at this bandwidth, with the estimated 2 interactions per hour, 400 terminals could be supported.

COMPARISON

Comparing the effect of the packet technology with the same terminal operating with preassigned Frequency or Time Division Multiplexed channels (ignoring the losses due to TDM sync bits or FDM guard bands) the circuit oriented terminal would require a 40 bit/sec transmit channel and a 4KB receive channel if a 5 sec write time is to be achieved. For 400 terminals with a 5 sec write time, the circuit method would require a total of 16 Megabits/sec bandwidth whereas the packet method only requires 20 Kilobits/sec bandwidth. Thus, the circuit technology requires a factor of 800 more bandwidth than the packet technique. Of course, the circuit mode terminals could interact more often within the same bandwidth right up to continual rewrite of the display every five sec, but you would have to massively reshape the user statistics to suit the technology.

Another possibility, to design the terminal so that it performed more effectively in a circuit oriented mode, would be to put character generation logic and position logic in the terminal. This would considerably increase the cost of the terminal, which originally had very little logic except shift registers. The result of adding this logic, however, is to reduce the bandwidth by a factor of 10 to 1.6MB or still 80 times the packet technique. The same logic would help reduce the packet size but, in order to maintain the graphic output capability and gross simplicity, it does not seem to pay.

CONCLUSION

As can be seen from the example, packet technology is far superior to circuit technology, even on the simplest radio transmission level, so long as the ratio of peak bandwidth to average bandwidth is large. Most likely, the only feasible way to design a useful and economically attractive personal terminal is through some type of packet communication technology. Otherwise one is restricted to uselessly small numbers of terminals on one channel. This result may also apply to many other important developments, only to be discovered as the technology of packet communication is further developed.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
SJCC 1970

2 F E HEART R E KAHN S M ORNSTEIN W R CROWTHER D C WALDEN
The interface message processor for the ARPA network
SJCC 1970

3 L KLEINROCK
Analytic and simulation methods in computer network design
SJCC 1970

4 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
SJCC 1970

5 S CARR S CROCKER V CERF
HOST-HOST communication protocol in the ARPA network
SJCC 1970

6 L G ROBERTS
A forward look
Signal Vol XXV No 12 pp 77-81 August 1971

7 N ABRAMSON
THE ALOHA System—Another alternative for computer communications
AFIPS Conference Proceedings Vol 37 pp 281-285 November 1970

8 D C ENGELBART W K ENGLISH
A research center for augmenting human intellect
AFIPS Conference Proceedings Vol 33 p 397 1968

9 D C ENGELBART
Stanford Research Institute Menlo Park Calif (Personal communication)

10 W C LYNCH
Reliable full-duplex file transmission over half-duplex telephone lines
Communications of the ACM Vol 11 No 6 pp 407-410 June 1968

11 K A BARTLETT R A SCANTLEBURY P T WILKINSON
A note on reliable full-duplex transmission over half-duplex links
Communications of the ACM Vol 12 No 5 pp 260-261 May 1969

An overview of programming languages for specialized application areas

by J. E. SAMMET

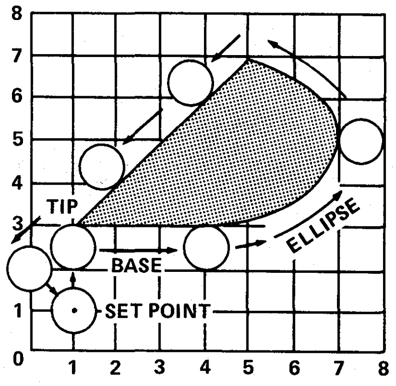
IBM Corporation
Cambridge, Massachusetts

INTRODUCTION

There are more than 165 different programming languages in use in the United States today, where only higher level languages are considered as programming languages. If assembly languages were considered in this total it would obviously be much higher. The total number would be still greater if programming languages in use outside the United States were included. (They are excluded here only because of the difficulty of obtaining accurate and sufficiently detailed information.) As individuals, and as an industry, we should ask ourselves, "What is the reason for this enormous proliferation?", particularly since many of these languages claim to be "general purpose." Some languages *do* serve a wide variety of users and applications, whereas others are restricted in intended usage. The languages which have few users are usually in that category because (a) they are basically for a narrow application area which has relatively few users* or (b) information about the language has not been widely disseminated, or (c) the language and/or its implementation is ineffective and/or has inadequate support, or (d) the language is implemented only on a computer not widely used. The purpose of this paper is to provide some of the background and perspective on the existence, classifications, and general characteristics of those languages which are oriented toward a specialized application area.

The earliest of the *major* "specialized languages" seems to be APT, developed at MIT by 1955, for numerical machine tool control. A small program—shown in Figure 1—illustrates in an intuitive way the type of language with which this paper is concerned. It is perhaps unfortunate—but is certainly quite true—that one of the major reasons for the proliferation of programming languages is that designing and implementing languages are fun, and there is a very large NIH (Not Invented Here) factor that makes even minor deficiencies in an existing language a justifiable cause for the development of a new one. This represents the less productive aspect of the proliferation. However, the important cases (meaning the languages widely used) fulfill a bona fide need for a language that can be used by people who don't really understand programming. The specialized languages help these people work in their own professional jargon. The motivation for the development usually comes when individuals find that for each existing language there are facilities that they want, and which they think the language should legitimately contain, but which are not basically available in the language. It is important to emphasize the "not basically available" aspect, as well as recognizing that there is a value judgment involved on this issue. There are certainly cases where FORTRAN has been used to write payroll programs but it is unlikely that anybody would seriously contend that such usage was appropriate; alternatively, people should not condemn FORTRAN for being ill-suited for writing data processing applications since that was not its intent. Similarly, COBOL has been used to generate differential equation programs, but that is certainly a perversion of its major intent. Thus, in considering whether an existing language should be used for a particular problem, its avowed intent must be kept well in mind. This applies not only to the syntax and semantics of the language itself, but also to the type of the machine or environment for which it was designed. A language suitable for use in a batch system is not necessarily well-suited for use in an interactive mode, even though a compiler or an interpreter for the language can indeed be put into a time sharing system. Similarly, a language which provides very good inter-

* Some languages which have a narrow area of intended usage may actually have a large number of users, e.g., COGO.



Part to be cut

Part Program	*Explanation*
CUTTER/1	Use a one inch diameter cutter.
TOLER/.005	Tolerance of cut is .005 inch.
FEDRAT/80	Use feedrate of 80 inches per minute.
HEAD/1	Use head number 1.
MODE/1	Operate tool in mode number 1.
SPINDL/2400	Turn on spindle. Set at 2400 rpm.
COOLNT/FLOOD	Turn on coolant. Use flood setting.
PT1 = POINT/4, 5	Define a reference point, PT1, as the point with coordinates (4, 5).
FROM/(SETPT = POINT/1, 1)	Start the tool from the point called SETPT, which is defined as the point with coordinates (1, 1).
INDIRP/(TIP = POINT/1, 3)	Aim the tool in the direction of the point called TIP, which is defined as the point with coordinates (1, 3).
BASE = LINE/TIP, AT ANGL, 0	Defined the line called BASE as the line through the point TIP which makes an angle of 0 degrees with the horizontal.
GOTO/BASE	Go to the line BASE.
TL RGT, GORGT/BASE	With the tool on the right, go right along the line BASE.
GOFWD/(ELLIPS/ CENTER, PT1, 3, 2, 0)	Go forward along the ellipse with center at PT1, semi-major axis = 3, semi-minor axis = 2, and major axis making an angle of 0 degrees with the horizontal.
GOLFT/(LINE/2, 4, 1, 3,), PAST, BASE	Go left along the line joining the points (2, 4) and (1, 3) past the line BASE.
GOTO/SETPT	Go to the point SETPT in a straight line.
COOLNT/OFF	Turn off coolant flow.
SPINDL/OFF	Turn off spindle.
END	This is the end of the machine control unit operation,
FINI	and the finish of the part program.

Source: Hori, S. *Automatically Programmed Tools,* Armour Research Foundation of Illinois Institute of Technology, AZ-240, Nov. 1962.

Figure 1—APT (machine tool control)

active facilities does not necessarily provide the broad flexibility and control normally found (or needed) in batch programs of great complexity.

In order to more explicitly establish the level of language that is being discussed, the defining characteristics of a higher level language are considered to be the following: (1) machine code knowledge is unnecessary, i.e., the user does not have to know the machine instructions available on any computer; (2) there must be good potential for converting a program written in a higher level language onto another machine; that is equivalent to saying that the language must be basically machine-independent (but since we know that no languages to date are *completely* machine-independent, what is being stipulated is a good potential for conversion to another computer); (3) there must be an instruction explosion, i.e., for most of the statements that the user writes in the higher level language, the computer (through a compilation or interpretive process) should generate many more machine instructions; (4) the notation of the higher level language should be more problem-oriented than that of an assembly language, i.e., it should be more "natural" for the class of problems being solved.

MEANING OF, AND STATISTICS ON, APPLICATION-ORIENTED LANGUAGES

Terminology

Many people refer to special-application-oriented languages as "special purpose" languages. This is misleading since the term "special purpose" really applies to a single or a limited set of objectives. For example, one might design a language that was intended to be very easy to use (e.g., BASIC). On the other hand, a major objective of the language might be easy and rapid compilation (e.g., MAD). Alternatively, the objective might be a language in which it was easy to generate efficient object code. Finally, as a particular type of "special purpose," a language might be designed to be useful for a specific application area.

Another term which is frequently used for the class of languages discussed in this paper is "problem-oriented." This is bad terminology, because *all* programming languages are problem-oriented. The main distinctions pertain to the width or narrowness of the problem area involved.

In the most fundamental sense, *all* programming languages are "application-oriented." Every programming language which has been developed has been aimed at a particular class of applications, which may be narrow or broad. In the latter case, the so-called general purpose languages such as PL/I and ALGOL 68 attempt to be suitable for *every* application area and they clearly don't succeed. This is not surprising when one considers the enormous variety of uses to which computers are put today, including calculation of space trajectories, medical diagnosis, inventory control, numerical machine tool control, graphic displays, and movie animation. With the sole exception of medical diagnosis, all of these applications have fairly general (e.g., FORTRAN, COBOL) or very specialized (e.g., APT) languages used for their solutions. Hence, it is inaccurate to refer to some programming languages as "application-oriented" while others are not. *All* programming languages are application-oriented, and the only question that can meaningfully be asked is "what type of application?". The answer can be narrow or broad, and also can be classified by either technique or application area, or both. For example, matrix manipulation is both a technique and a class of applications, depending on your viewpoint. (This is of course analogous to the concept that one person's program is somebody else's subroutine.)

In contrast with the technique and/or application of mathematics, one might consider the application area of technical computation with engineering as a major subcategory; underneath the latter can be logical design and civil engineering (which in turn has subdivisions of structural engineering and coordinate geometry). In parallel with engineering we might have astronomy. The area of computations in medicine could be considered a subset of technical computations, or might be considered a major category. It is really up to the individual as to how fine a line he wants to draw. Table I represents one possible schematic, and the reader can easily develop similar tables to reflect his own approach to problem areas. It is important to note

TABLE I—A Schematic for Types of Application Areas

- Mathematics
  - Numeric
    - Matrices
    - Partial Differential Equations
  - Non-numeric
    - Matrices
    - Partial Differential Equations
    - Polynomials
- Technical
  - Engineering
    - Logical Design
    - Civil Engineering
      - Structural Engineering
      - Coordinate Geometry
  - Astronomy
- Medical

that the distance of the observer from an application area affects his view of it. Thus, civil engineering is one application area from the viewpoint of a computer scientist, but has many levels and subdivisions for a person involved in building bridges.

Another method of classifying the application areas is through the realization that some types of applications involve a particular specialized discipline or specific technology, whereas others are specialized but can be used across many areas. Illustrations of the former include civil engineering, movie animation, social science, etc., whereas the latter include simulation, graphics, etc. A more complete list is given in Table II.

This paper specifically deals with, and includes in the category of "languages for specialized application areas," all programming languages *except* those specifically intended for mathematical computations (both numeric and non-numeric, i.e., formula manipulation), business data processing, string and list processing, or any combinations of these. (The use of a language outside its intended application area does *not* cause it to be considered specialized, nor change the definition of its primary intended use.) This list of excluded categories is certainly debatable on the grounds of what it does and does not contain. The applications cited seem so fundamental to the use of computers that none should be considered as being specialized. Of the categories of specialization (discussed later) only simulation seems to have a reasonable potential for consideration as fundamental, and at this point in time such a conclusion does not seem justified to the author. Note also that systems programming is considered one of the specialized application areas; this view refutes the concept which has been expressed by some people that a language can only be considered good if it can be used to write its own compiler.

The terms "special-application-languages" and "application-oriented languages" will be used in this paper as synonyms for the phrase "languages for specialized application areas."

TABLE II—Categories and Statistics on Languages for Special Application Areas

	Number of Languages					
	Pre 1966			1966–1971		
	1–2	3–5	over 5	1–2	3–5	over 5
Specialized Application Disciplines						
automated equipment checkout	X					X
civil engineering	X					X
computer assisted instruction			X			X
logical design (including simulation of digital computers)			X		X	
machine tool control		X				X
movie animation		X			X	
real time/process control		X				X
social science/humanities	X				X	
systems programming			X			X
Narrow Applications Across MultiDisciplines						
computer aided design	X			X		
graphic and on-line display		X				X
query (excluding data base management systems)			X		X	
simulation—continuous (including block diagrams and analogue computers)		X				X
simulation—discrete			X			X

Specific application areas and statistical trends

In Table II is a list of special application areas which have had programming languages developed for them. Some statistics are also given; the latter are presented in categories rather than specific numbers in order to show trends. It is virtually impossible to obtain a completely accurate count because of the existence of dialects, multigeneration languages (e.g., SIMSCRIPT I, I.5, II, II.5, and II Plus), and the differences between dates of definition, implementation, and actual usage. The distinction between pre-1966 and more recent years largely reflects existence on second generation computers. However, not all languages on second generation computers were implemented on third generation machines. The most complete lists of languages available—although they are by no means perfect or complete—are in References 15, 16, 17, 18. Discussions of almost all those languages developed by 1967 are in Reference 14, Chapter IX.

General statistical trends

As indicated just above, it is virtually impossible to obtain accurate statistics. This is due not only to the language generation problem but also to the fallibility of any single human attempting to track this field. Until we get some scientific methodology for defining language generations, dialects, etc., and until a reliable reporting system is developed, the most we can hope for is some statistical trends. (See Reference 19, for a first attempt at dealing with the dialect problem.)

The trends that can be observed are all based on References 15, 16, 17, 18. While the figures cited have inaccuracies and in particular are based on date of inclusion of a language in the roster rather than its actual development, it seems fair to state that all the errors are probably consistent and hence relatively unbiased.

The first year in which any serious attempt was made to list "all" the special-application-oriented languages was 1968. In each of the four years 1968-1971 the number of special application languages was approximately 50 percent of the total number of languages listed. The latter were 82, 125, 139, and 164 in the four cited years. In 1969, 1970 and 1971 the *total* number of new languages added from the previous year was between 35 and 40 each year. In 1970 and 1971 the percentage of new special-application-languages was between 40 and 60 percent of all the new ones.* In both 1970 and 1971 the number of special-application-languages dropped from the roster (meaning clear non-use of the language) was about 50 percent of the total number of languages dropped. (The latter numbers are small, ranging between 5 and 10 percent of the total.)

These numbers are presented because of the trends they indicate, and certainly should not be used as definitive representations of the actual situation. For that reason these figures were deliberately *not* presented in a table or graph because that would imply more accuracy than is justified.

NEEDS TO BE MET IN DEVELOPING THE LANGUAGES

There are several needs which must be met by a new language or a modification of an existing one. The major needs are functional capability, a style suitable fcr the application area, and efficiency.

Functional capability

The inclusion of specific functional capability is probably the single most important need to be met by the application-oriented languages. Most of the deficiencies in a particular language actually reflect the omission of functional capabilities which a certain class of users need. For example, the FORTRAN user often wishes to do character manipulation, the COBOL user might wish to do floating point calculations, and the PL/I user might wish to do simulation. In considering functional capability, the three major issues are (a) the specific features which must be included (i.e., commands and data types), (b) the distinction between subroutines and a specific language facility, and (c) the facilities provided by the system. Each of these is now discussed in more detail.

Specific features

The specific features representing the functional capability of *any* language are the operations (i.e., commands) to be performed, and the data types. The kinds of operations to be provided depend completely on the application area, and of course must be consistent with the data types. As a very simple example, the existence of floating point numbers is extremely important in any language to be used for scientific computation, whereas they have little or no importance for business data processing. Therefore, the existence of a floating point or complex number data type is a functional requirement of many scientific problems, and naturally there must be executable statements which can operate on these data types. A less elementary example occurs in the civil engineering field where the ability to calculate the point of intersection of the tangents to two circles may be deemed important enough to incorporate directly into the language. Furthermore, a "point" with coordinates might be defined as a data type with operations provided to calculate areas. In the case of an application involving simulation, some indication of time and time-dependencies in both the data types and commands is absolutely crucial. In the case of a language developed for animated movies, the user must be able to address and manipulate pointers or markers on a screen. In the general area of graphics, a "graphical" data type is normally needed.

Subroutine versus language facility

Since all major languages have a subroutine capability of one kind or another, the question can legitimately be asked as to why the needs cannot simply be met through adding subroutines rather than either developing a new language or adding syntax and se-

* A similar figure for 1969 would be very misleading because the 1968 Roster did not contain any of the languages developed for CAI and their inclusion in 1969 perverts the statistics.

mantics to the existing language. It must be emphasized very strongly that there is a fundamental difference between functional capability (which can be represented either directly in the languages *or* via subroutines) and specific language facilities.

The easiest way to show this is by means of an example within the field of mathematics. (Since this was ruled out earlier as not being specialized, its use here is primarily to illustrate the concept of subroutines versus specific language facilities.) Suppose a person wished to do MATRIX addition in a language like FORTRAN. One could add a subroutine to perform those calculations and then invoke this facility by writing

```
CALL MATADD (A, B, C)
```

This is fundamentally different in several ways from writing something of the following kind:

```
DECLARE A, B, C MATRICES
C=A+B
```

In the first instance, the user (meaning both the writer and reader of the program) has lost a certain mnemonic advantage and furthermore has the difficulty of remembering the types, sequence, and restrictions on the parameters that he is using. In this particular example, the main difficulty is to remember whether the stated call would provide A as the sum of B and C, or C as the sum of A and B; this is something which must be remembered or looked up in a manual. In the second example, what is happening is far clearer. In addition to the lack of "problem-oriented notation" inherent in using the CALL or equivalent statement, there are implementation difficulties and inefficiencies relative to linkages. In many cases, there is a great deal of extra code generated to invoke a subroutine, although the latter may in fact be quite small and would be more efficient placed in line. Unless the compiler is very sophisticated, it is likely to generate the linkage coding rather than placing the subroutine in line.

While subroutines do serve a very useful purpose in making additional functional capability available to the user, they should not be viewed as substitutes for additions to a language.

System provided facilities

Another aspect of functional capability is the built-in facility of the language and its associated compiler to do things which might need to be programmed in another language. For example, in the languages for computer assisted instruction, it is assumed that they will be used in an interactive mode and hence the language translators automatically provide for input and output from a terminal without the programmer having to specify much (if anything) about input/output. Furthermore, certain types of control flow are assumed by the system and handled automatically. In the case of graphic languages, the language (and its processor) automatically provide the necessary instructions to the graphics equipment.

Style suitable for application area

In the development of languages for special application areas, the style plays a major role, primarily because the users are normally not professional programmers. Style in a programming language has many facets, ranging from personal views on the importance (or non-importance) of blanks and punctuation, to the use(or non-use) of a GOTO command, to the selection of a particular word for getting input data (e.g., READ vs. GET). The major identifiable elements of style are vocabulary, personal preferences, and requirements affecting style.

Vocabulary

The second* most important need in an application-oriented language is the vocabulary or professional jargon that is involved. The whole *raison d'etre* of many languages for specialized application areas is to allow the user to write his specialized jargon in a natural manner. Whereas he can certainly develop subroutines to accomplish what he wants, the ability to use his own terminology in a style natural to him is of paramount importance. It is normal that a person outside a particular area will find the nomenclature confusing or not understandable. All figures in this paper reflect this issue, i.e., the programs as written are generally not understandable to any reader outside the specific application area.

Personal preferences

It is unfortunate—but quite true—that in the development of one of these special-application-languages, the personal preferences of the developer have a significant effect, although they should of course be subordinated to the functional requirements and the vocabulary. For example, people who wish to have short data names and short statements in a language because

* The first is the functional capability.

they like that style may be forced into a different mode because the professional jargon of the field constantly uses long words. (In some systems, e.g., COGO, both short and long forms of key words are allowed to provide short cuts for experienced users.) The choice of a fairly rigid format versus a free form, or the selection of specific key words (e.g., READ versus GET) can sometimes make the difference between successful usage of the language versus constant unhappiness by the users. In some instances, people have strong views on punctuation and will change an existing language to eliminate (or include) punctuation in lieu of some other syntactic restriction. (It has been said facetiously that by choosing the correct subset of PL/I one merely has FORTRAN with semicolons.)

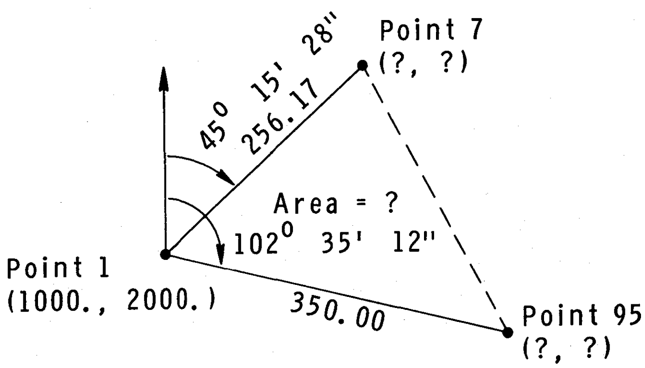
Background and past experience with specific equipment often have a strong personal influence. People who have used punched cards extensively tend to favor rigid formats with card columns used as syntactic delimiters. Those who have used on-line terminals generally tend to favor free form. Even in this latter case, there is considerable difference of opinion on the value of limiting each statement to the length of a single line.

The use of very simplistic styles, as illustrated in Figure 2, primarily consisting of a single key word at the beginning of a line, followed by some parameters, certainly forces one to reconsider the border line between programming languages and powerful macros. Certainly such a style is generally not "problem-oriented" which was described as one of the characteristics of a higher level language. However, such languages (e.g., COGO—see Figure 2) can be justified as higher level languages because of the distance of these operations from normal machine code. Thus, there is a large amount of information built into the compiler or the interpreter to perform specific functions which are directly related to an application (rather than merely enhancing normal machine code).

It should be emphasized that there is little or no way of determining which of two styles is better; in virtually every case it is a matter of individual taste, preference, previous usage, and jargon common to the application area.

Requirements affecting style

It should not be thought that *all* matters of style are arbitrary; some are influenced or forced by specific requirements of equipment—particularly the character set. In other cases, the intended use may affect the style of the language and environment will have a very specific effect. If the language is to be used in an interactive mode, it is pointless to be highly concerned about card columns. On the other hand, if the primary use of the language involves relatively simple statements and a large number of numeric parameters or data, then it may be most effective to orient the style of the language toward card columns.



```
STORE                 1     1000.       2000.
LOCATE/AZIMUTH   7    1      256.17     45 15 28
LOCATE/AZIMUTH  95    1      350.00    102 35 12
AREA                  1  7    95
PAUSE
```

Explanation:

Small COGO program for figure shown. In the figure above, given the coordinates of point 1, the length and azimuth (clockwise angle from north) of lines 1–7 and 1–95, the COGO program shown computes the coordinates of points 7 and 95 and the area of the triangle. In the program, the second line reads: Locate point 7 by going from point 1 a distance of 256.17 at an azimuth of 45 degrees 15 minutes 28 seconds.

Source: Fenves, S. J. "Problem-Oriented Languages for Man-Machine Communication in Engineering," p. 48. Reprinted by permission from *Proceedings of the IBM Scientific Computing Symposium on Man-Machine Communications*, Data Processing Division, 320–1941, © 1966 by International Business Machines Corporation.

Figure 2—COGO (civil engineering)

In some instances, the style can be justified on fairly concrete grounds. For example, in situations where time is critical (e.g., command and control) there may be much more need for brief fixed formats with little flexibility. In other areas where documentation plays an important role, the desire may be for lengthy free form syntax which clearly conveys the meaning of the program to a large number of persons long after the program is written.

Efficiency

It is obvious that one of the reasons for developing these application-oriented languages is efficiency. This involves the efficiency which comes both from ease of writing the programs and from suitable processors.

One major way in which an attempt is made to increase efficiency is to delete unwanted parts of a major language while (perhaps) adding on the functional capability (in the form of new language elements). While the person or group which has a large language available to them is under no obligation to use all of its facilities, it is perfectly clear that they are paying for the implementation of these unwanted features. The common and obvious practical solution to this problem is merely to have a compiler which implements a subset of a language, and this is frequently done. However, to the extent that the individual or group wants to clearly define a certain subset of syntax and semantics and give it a name, he has in effect defined a new language. It is *not* true that all languages can have subsets properly defined, if a program written in the subset is also to run on a compiler for the full language.

The ways in which suitable processors can be obtained are discussed in the next section.

DESIGN PARAMETERS OF SPECIAL APPLICATION LANGUAGES

The design parameters of the languages for special application areas essentially reflect the needs which were discussed in the previous section. Thus, the functional capability, a style suitable for the application area, and efficiency are clearly design parameters. However, there are two additional issues which were not discussed in the previous section but which are very significant in the development of these languages, namely the methods actually used to design and implement the languages. It is well-known that those issues are important in the design of any language, but they are perhaps more significant in the languages within the scope of this paper because the intended users are not professional programmers and hence are less likely to be tolerant of unnecessary idiosyncrasies. In more general languages, some of the idiosyncrasies are (unfortunately) forced into existence by the methodology used for language design and/or implementation. Finally, it is possible to summarize the potential language requirements.

Methods of defining language

There are three basic methods for defining a language. The first is the most obvious—namely, a straightforward definition. In this instance, the individual or group merely sits down and writes out a definition of the language. Depending upon their sophistication and the complexity of the language, they may use something like Backus Normal Form, or alternatively may simply use ordinary English and examples. A special case of this method involves making specific changes to an existing language, which may involve addition, deletion, changes, or all of these.

The second method of defining a language is through an extensible language system. This is an area which has become increasingly important over the past few years, although there is not much evidence of practical systems or significant usage as yet (see Reference 1). In this situation, the developer of the language for a special application area is limited in two ways. First, he is limited by the inherent style and capability of the base language, and second, he is constrained by the mechanism given to him to produce the extensions. If the extension facilities do not allow new data types to be added, then he is limited to the syntax and functional operations of new commands. For example, any macro system (e.g., PL/I) tends to allow new commands and/or new syntax to be developed but does not provide for new data types. Alternatively, other extensible systems, e.g., IMP[8] allow for both new commands and new data types, but do not allow for major changes in language style or format.

The third method of defining the language is via some system. In this case, which seems to be the most important and the most promising, the user or application programmer states his wishes to a person who can be defined as a "language designer" who then interacts with a system which produces *relatively easily* a language definition which meets the needs of the original user or programmer. While many people have talked about this for years, relatively little has actually been accomplished—see a later section. In the long run, it is clear to me that we must allow the user ample facilities to easily define and implement his own language subject to whatever constraints and quirks he may have. The key word here is "easily" and that is the major difficulty in achieving the general goal.

Methods of implementation

Just as there are several different ways of defining a language, so there are different broad techniques for implementing them, and to some extent (but not entirely) they match the methods of defining the language. The first and most obvious method of implementation is a specific compiler or interpreter. This would tend to be used most often in a case where a language had been designed from scratch. Second, paralleling exactly the use of extensible languages is the extensible language compiler or equivalent facility. This method might conceivably be used with a language designed in another way, but it is highly unlikely to be applicable. A third possibility is a very powerful macro assembler which then allows the user quite a bit of flexibility in terms of jargon, lists of parameters, etc., but gives him virtually no flexibility in style and overall format. Finally, roughly corresponding to the user-defined language via a system, is a system which generates a compiler or interpreter. This method of implementation can be used even with the first case where the individual has designed and defined the language from scratch. The compiler generators that have been the vogue for many years come close to satisfying this requirement in theory, although none seem to do it in practice. (See a later section.)

TABLE III—Considerations in Language Design

Syntax

1. Form
   - Free
   - Fixed tabular, as in report writers or decision tables
   - Rigid with required parameters, i.e., essentially macro style
2. Punctuation
   - Critical with many symbols or not very significant
   - Blank character is critical delimiter or not significant
   - Single or multicharacter
3. Identifiers and Key Words
   - Single or few characters vs. lengthy
   - Abbreviations optional vs. not allowed
   - Specialized vocabulary
4. Full power of a language like PL/I (for systems programming)
5. Declarations—see under Data

Data

1. Types
   - Specialized to application, e.g., length
   - Many vs. few
   - Implicit vs. specifically declared
   - Determined by position in statement
2. Declarations
   - Grouped by attribute vs. grouped by data item
   - Implicit from other syntactic rules
   - Default conditions

Semantics

1. Specialized computational routines
2. Non-standard use of common characters (e.g., plus sign)

Program/Control Structure

1. Generally quite simple with no block structure nor compound statements
2. Very powerful (for systems programming)

Potential language requirements

A brief summary of requirements (or considerations) for potential desired language features is shown in Table III. Obviously, any one language will only use some of these. However, it is possible to find at least one specialized language which requires or uses each of these approaches or features.

SOME SPECIFIC APPLICATION AREAS WITH EXAMPLES

Almost all special application areas tend to look small and homogeneous when viewed from outside, but large and filled with differing problems and issues when

```
qu Who discovered America?
aa Ericson
ab Leif Ericson
ty Your answer would be accepted by some.
ca Columbus
ty Yes
wa Ponce de Leon
ty No. He looked for the "Fountain of Youth."
   Try again.
un bl
```

Explanation:

Example uses the aa and ab operation codes. If the student enters a response of "Ericson," or "Leif Ericson," the message "Your answer would be accepted by some," is typed to the student. After the aa or ab match, the system continues to scan statements until it finds the un statement. It then types the contents of buffer 1. If the student responds with an answer which does not match an aa, ab, ca, or wa statement, only the un argument (the contents of buffer 1) is typed to the student. The contents of buffer 1 might be, "Please try again." Using a buffer in this way saves the author from repeatedly entering the same text for many un statements in the course.

Source: Reprinted by permission from p. 17, *Coursewriter III for System/360, Version 2, Application Description Manual,* Data Processing Division, H20-0587, © 1969 by International Business Machines Corporation.

Figure 3—COURSEWRITER III
(computer assisted instruction)

viewed by those familiar with the field. A very good illustration of this can be seen merely by glancing at the papers on numerical control programming languages.[10] Even though one language—namely APT—predominates, there are still enough technical issues surrounding its utility and implementation to cause the existence of numerous other languages. In particular, 32 others are listed.[11]

In the field of computer-assisted instruction, there are over 30* languages utilizing different syntactic and conceptual approaches. Just glancing at Figures 3 and 4, for Coursewriter and FOIL, respectively, shows the wide variety of style, ranging from two-letter mnemonics to a PL/I-like language. A comparison of CAI languages can be found in Reference 23.

The situation in graphics is similar, but with another dimension of concept involved. In the computer-assisted instruction application area it can reasonably be argued that the application is unique and that there is little relevance to existing languages of somewhat wider purpose, e.g., FORTRAN, COBOL. (However, even in the CAI situation, the case can be made for the desirability of control statements, conditionals, and numeric calculations expressable by formulas as in FORTRAN.) In the field of graphic languages there is certainly a special facility that can be used by many applications (including, for example, computer-assisted instruction). The technical approaches and issues in graphics are as diverse as in other applications. In this case more of an argument can be made for providing the facilities as extensions to existing languages, e.g., GRAF,[5] which is an extension of FORTRAN. However, most developers went to the other extreme with entire new languages, for example General Purpose Graphic Language.[9] (See Figures 5 and 6, respectively.) Some languages take a middle ground by retaining some of the style of more popular languages, such as ALGOL or FORTRAN, but by no means accept compatibility with them as a constraint. (See for example Euler-G[12].) In each case the developer of the language was reflecting his view of how graphic information should be stored internally, and the most effective way in which it could be displayed and manipulated on a screen. In this application area, the physical environment plays a major role in the development of the language; thus the existence of lightpens, keyboards, push-buttons, etc., must be supported—if they are to be used by the graphic language.

```
TY WOULD YOU LIKE TO CONTINUE THE
  EXERCISE?
ACCEPT
    IF 'NO', GO TO FINISH
    IF 'YES, OK'
      NUM=NUM+1
      GO TO NEXT
    GO BACK PLEASE ANSWER YES OR NO
```

Explanation:

The TY causes the indicated typeout to be made. If the student responds with a NO there is a branch to a statement labeled FINISH. If either YES or OK are typed in the variable NUM has 1 added to it and control is transferred to the statement labeled NEXT. Any answer not including YES, NO, or OK causes the typeout from the system of PLEASE ANSWER YES OR NO and a return of control to the ACCEPT statement.

Source: Hesselbart, J. C. "FOIL- A File-Oriented Interpretive Language," *Proceedings of the 23rd National Conference of the Association for Computing Machinery*, 1968, p. 94. © 1968 Association for Computing Machinery, Inc. Reprinted by permission.

Figure 4—FOIL (computer assisted instruction)

SYSTEMS FOR DEVELOPING LANGUAGES FOR SPECIAL APPLICATION AREAS

It is unfortunate, but appears to be a fact, that there are no currently available systems which have actually been used in a practical way for the development of a *significant number* of languages (and their processors) for special application areas. It is not even

```
DISPLAY A, B, PDQ, POLE (11), K72A (7, 2, 4)
PDQ=A+POINT (0, XY2)+B
K72A(2, 1, 3)=PLACE (0, 1)+PRINT 13,
  (YY(I), I=1, 8)+PLACE (100, 200)
```

Explanation:

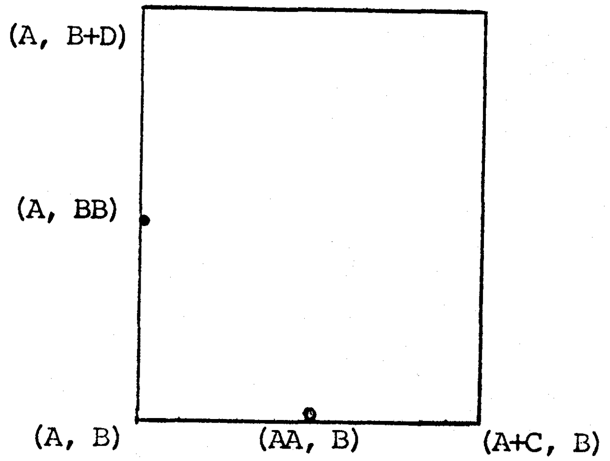
The names in the DISPLAY statement are display variables. PDQ is assigned the value which is obtained by first generating the graphic orders of A followed by the orders generated by the built-in function POINT, followed by the orders generated by B. Similarly, the value of K72A(2, 1, 3) is obtained by taking each of the graphic orders indicated in turn. The built-in display function POINT generates orders for plotting the point with indicated coordinates; the built-in function PLACE changes the beam position without plotting, and PRINT plots the indicated string of characters.

Source: Based on examples in "GRAF: Graphic Additions to FORTRAN," A. Hurwitz and J. P. Citron, *Proceedings of the Spring Joint Computer Conference*, 1967.

Figure 5—GRAF (addition to FORTRAN to do graphics)

* Not all of these are listed in Reference 18.

```
BEGIN WINDOW (A, B, C, D)
RECT (A, B, C, D)
AA=A+C/2
BB=B+D/2
LINE AA, B; AA, B+D
LINE A, BB; A+C, BB
END
```



Explanation:

A subroutine WINDOW is defined where A and B are the coordinates of one corner of a rectangle and C and D represent the horizontal and vertical dimensions. The subroutine to draw a rectangle is called and executed. The drawings of these windows are to have horizontal and vertical lines midway in the window so AA and BB compute the coordinates of the midpoints. The LINE commands cause the "midway" lines to be drawn.

Source: Kulsrud, H. E. "A General Purpose Graphic Language," *Communications of the ACM*, Vol. 11, No. 4, April 1968, p. 253. © 1968 Association for Computing Machinery, Inc. Reprinted by permission.

Figure 6—General purpose graphic language

clear that any of the systems now in prototype stage will ever be satisfactory for that purpose. Virtually all of the systems known to the author have one or more of the following characteristics: (1) they require a compiler expert for their use; (2) they have been used to produce some minor variations on "normal" languages such as FORTRAN, ALGOL, etc.; (3) they are not *really* intended to be used to develop the types of languages discussed in this paper; (4) they give lip service—although little else—to the concept of allowing the average user to be able to define his own language and easily get a processor for it.

In theory, any compiler-compiler, meta-compiler or similarly designated system could be used for this purpose. However, there is a different emphasis in most of those developed to date. They have been designed primarily to provide an easy means of implementing known and widely used languages (e.g., FORTRAN, COBOL, ALGOL, PL/I) rather than as a tool for the development of new languages with uncommon requirements, and their processors. Thus the major considerations have pertained to efficiency of the resulting compiler, with an easy way to make minor changes in the syntax. A discussion of the past and potential use of such systems or translator writing systems in general is beyond the scope of this paper. A good survey is given in Reference 3.

Although no systems seem to have been widely, or even significantly, used for developing the types of languages within this paper, several have had limited use and/or have such intent for the future. A brief description of these will now be given.

(a) ICES

The Integrated Civil Engineering System (ICES) provides an overall system within which many language processors suitable for civil engineering can reside and use common facilities.[13] There is also the capability of allowing the user to define new languages, or add facilities to one of the existing languages. This is done by means of the Command Definition Language (CDL). Although CDL has not been used very much in practice, at least one language, namely STRUDL,[22] was developed using it. (A brief but relatively accessible summary of ICES, including CDL, is in Reference 14.)

(b) REL

The Rapidly Extensible Language (REL) System was (and is) intended for use by researchers in the fields of complex social and environmental processes.[20]

It has a powerful English grammar, thus permitting individuals to communicate with the computer in a fairly "natural" language. In 1970 an experimental system was in operation on the IBM System 360/50 and was used to develop an animated film language and also by some social scientists.

(c) PLAN

The Problem Language ANalyzer (PLAN) has many facets to it, but the only one of interest in this context is its facility to allow the user to define simple new languages in an easy manner.[6] A version providing graphics support allows the user to develop his language at an IBM 2250 and also provides him with many built-in graphics facilities.[7]

(d) UAL

The User Adaptive Language (UAL) is another attempt to provide a user with the ability to dynamically create and modify a language in an interactive mode.[4] This system provides the user with fairly sophisticated programming concepts (e.g., lists), but does not require him to use them.

(e) SDF

The Syntax Definition Facility (SDF) allows the user to define his language by means of illustrative sentences.[2] The system indicates whether the input is ambiguous or contradictory to earlier information. By late 1971, it had been used primarily to implement the syntax of fairly standard language subunits, e.g., arithmetic and Boolean expressions.

(f) Extensible Languages

All extensible languages should—in theory—be usable for creating specialized languages. By late 1971, none seem to have been used in this way. See Reference 1.

BRIEF COMMENTS ON THE FUTURE

It seems unfortunate but true that the proliferation of higher level languages is likely to continue at about the same rate. The reason for this is that some of the causes and motivations behind the development of these languages rest in quirks of human nature rather than technological progress or lack thereof. Thus, as long as people find it fun to develop languages, as long as they want something which is specifically tailored exactly to their needs, and as long as they are going to find picayune faults with the existing languages, there is very little that technical progress can do to reduce the number of languages. On the other hand, there are some areas in which improved technology will have an enormous effect. For example, the existence of good extensible language systems, or good systems which can easily generate a language and its translator based on appropriate input from a language designer, will have a considerable effect. We might even envision specialized language generators, i.e., a system designed to allow the easy creation and implementation of languages in a single application area, e.g., CAI, graphics. ICES[13] is a simple attempt in this direction for civil engineering.

In the opinion of this author, the ease and efficiency of using a language particularly suited to a specific application area is a desirable result which outweighs the disadvantages of the proliferation. A thorough discussion on the pressures and resources involved in the future development of these specialized languages is given in Reference 20.

SUMMARY

This paper has defined and discussed the class of languages which are designed for use in specialized application areas. These languages include about half of all higher level languages used in the United States at the time of this writing. A discussion of terminology showed why some of the commonly used terms for this class of languages are technically inappropriate.

The major needs to be met in developing these languages were shown to be functional capability, deletion of parts of an existing language, and a suitable style. Two specific application areas, namely CAI and graphics, were used to illustrate the existence of significantly different language styles within the same application area. Although there are a number of systems which purport to allow the user to easily define and implement his own language, and they are mentioned, none have actually been significantly used. A few brief comments on the future indicate that the proliferation serves a useful purpose and will continue.

REFERENCES

1 *ACM SIGPLAN*
Proceedings of ACM SIGPLAN Conference on Extensible Languages
SIGPLAN Notices Vol 6 No 12 December 1971

2 R S EANES
An interactive syntax definition facility
MIT Electronic Systems Laboratory Report ESL-R-397 September 1969

3 J FELDMAN D GRIES
Translator writing systems
Communications of the ACM Vol 11 No 2 February 1968

4 A HORMANN A LEAL D CRANDELL
User adaptive language (UAL): A step toward man-machine synergism
System Development Corporation Technical Memorandum TM 4539 June 1971

5 A HURWITZ J P CITRON J B YEATON
GRAF: Graphic additions to FORTRAN
Proceedings of the Fall Joint Computer Conference Vol 30 1967

6 *Problem language analyzer (PLAN) program description manual*
IBM Corporation Form GH20-0594

7 *PLAN graphics support for the IBM 2250 (application description manual)*
IBM Corporation Form H20-0535

8 E T IRONS
Experience with an extensible language
Communications of the ACM Vol 13 No 1 January 1970

9 H E KULSRUD
A general purpose graphic language
Communications of the ACM Vol 11 No 4 April 1968

10 W H P LESLIE Ed
Numerical control programming languages
North-Holland Publishing Company 1970

11 W E MANGOLD
Status of NC language standardization in I.S.O. (International Standards Organization)
Numerical Control Programming Languages W H P Leslie Ed North-Holland Publishing Company 1970

12 W M NEWMAN
Display procedures
Communications of the ACM Vol 14 No 10 October 1971

13 D ROOS Ed
ICES system: General description
MIT Dept of Civil Engineering R67–49 September 1967

14 J E SAMMET
Programming languages: History and fundamentals
Prentice–Hall Inc 1969

15 J E SAMMET
Roster of programming languages, 1968
Computers and Automation Vol 17 No 6 June 1968

16 J E SAMMET
Roster of programming languages, 1969
Computers and Automation Vol 18 No 7 June 1969

17 J E SAMMET
Roster of programming languages, 1970
Computers and Automation Vol 19 No 6B November 1970

18 J E SAMMET
Roster of programming languages, 1971
Computers and Automation Vol 20 No 6B June 1971

19 J E SAMMET
Problems in, and a pragmatic approach to, programming language measurement
Proceedings of the Fall Joint Computer Conference Vol 39 1971

20 F B THOMPSON *et al*
REL: A rapidly extensible language system
Proceedings of the 24th ACM National Conference 1969

21 F B THOMPSON B H DOSTERT
The future of specialized languages
Proceedings of the Spring Joint Computer Conference Vol 40 1972

22 R A WALTER
A system for the generation of problem-oriented languages
Proceedings of 5th National Conference Computer Society of Canada May–June 1966

23 K L ZINN
A comparative study of languages for programming interactive use of computers in instruction
University of Michigan Center for Research on Learning and Teaching February 1969

The future of specialized languages

by F. B. THOMPSON and B. H. DOSTERT

California Institute of Technology
Pasadena, California

INTRODUCTION

The prediction of the future, like the review of the past, has as its sole usefulness, the organization and purposeful direction of the present. Thus, in situations where there is rapid change, as there certainly is in the field of computing, prediction—though more difficult—is all the more necessary. Suppose that you, as a computer specialist, are considering whether to design and implement a language for your own special application area or to consolidate your programming within the framework of a single, general language. You will certainly want your decision to be responsive to, though perhaps not dictated by, your perception of the directions computers and computing will be taking.

Our own perception concerning programming language development rests on two considerations. The first consideration is to understand the nature of the pressures that will bring about change. New languages for specialized application areas don't just happen; they arise because of a felt need, a dissatisfaction with trying to write programs for a specialized domain using programming languages not specifically tailored to that domain. Our first task, then, is to understand the essential nature of this need.

The second consideration on which our perception of future developments rests is to assess the technological and economic restraints and resources for change. Big computers, batch processing, and further professionalization of programmers will stimulate certain trends in programming language design; mini-computers, interactive processing, and widespread knowledge of programming will stimulate different trends. Thus, our second task will be an assessment of the major developments in hardware and software, and of the economic environment within which the pressures for change will be resolved.

These considerations should then put us into a position where we can bring into some focus what to expect in language development for specialized application areas. The past and current situation has been reviewed by Jean Sammet in *An Overview of Programming Languages for Specialized Application Areas.*[19] Our final task, therefore, is to identify the major trends these developments are likely to take in the future.

SOURCES OF PRESSURE FOR LANGUAGE DEVELOPMENT

In talking about languages for computing, we should first like to consider certain aspects of the nature of language. To do this it will be useful to develop an analogy between a language, on the one hand, and the design engineer's laboratory, on the other. Imagine yourself to be an electrical engineer designing a new computer central processing unit. As your design progresses, you and your team actually construct a model or "breadboard" out of flip-flops, amplifiers, "and" gates and other components. You wire these together often using clips rather than solder, because as you try out your design, you may wish to modify it as your breadboard model reveals unexpected relationships and possibilities for improvement. This model has a second important function: it expresses in a most articulate way the sum total of ideas contributed by the various members of the design team and is thus a vehicle for communication between them.

The design engineer's laboratory provides a great variety of basic components and the means of hooking them together into meaningful arrays. As the design proceeds, selection is made of certain components and they are built into combinations that express the functions that the designer wishes to execute. The same is true of a language. A language is based upon a great variety of basic concepts and its syntax is the means of hooking them together into meaningful arrays. As a programmer proceeds, he selects certain components, building them into combinations that express the functions that he wishes to execute. This then is the analogy between design laboratory and programming language we now wish to exploit.

Suppose you were called upon, as a computer specialist, to develop the configuration of a new computer for your installation. You would have to choose the main CPU, the size of high speed memory, what peripherals you would want, etc. Suppose you were told that the components you had at your disposal were diodes, resistors, capacitors and the like and that you had to build your computer installation from this point up. You would surely go about your job far differently than if you had been provided as building blocks the major pieces of computer equipment available from computer manufacturers. Consider these two situations carefully. If you had diodes and resistors you would have to be a different kind of engineer than if you had disk drives and printers to work with; your design would take considerably longer, and your expectation, design considerations and final output would be distinctly different. Our activities, the direction of our thinking, our efficiency, and our results are all greatly influenced by the basic components with which we must work and the means at our disposal for building them into more meaningful structures.

These same observations apply with equal force to the languages we use in dealing with our various problem areas. Language is the principal tool of our conceptual processes. And different styles of thought reflect and are reflected in the languages we use. Notice the correlation between artificial intelligence and LISP; engineering mathematics and FORTRAN or PL/I; operating systems programming and assembly language. This notion that language conditions the modes of thought is not new. Benjamin Whorf, the MIT chemist-turned-linguist, expresses the matter in the following words:

> *"We dissect nature along lines laid down by our native languages. The categories and types that we isolate from the world of phenomena we do not find there because they stare every observer in the face; on the contrary, the world is presented in a kaleidoscopic flux of impressions which has to be organized by our minds—and this means largely by the linguistic systems in our minds."* [1]

Whorf was referring, of course, to natural languages. But surely there is every reason to suspect that the programming languages we use will also have an effect on the kinds of problems and the character of the programs to which we apply the computer.

To illustrate this interdependence, consider the relation between languages and the kinds of management information systems that are currently in vogue. The original development of concepts for dealing with business data came at a time when punch cards and magnetic tape were the only large size repository for machine readable data. Thus the whole nomenclature of sequential files, fields, records, file update, report generation, etc., became an integral part of the language of data processing and the conceptual environment for the development of COBOL. Now we find that the inertia resulting from these common language concepts inhibits the acceptance of new systems based upon random access to data and time shared environments. A whole generation of programmers, information systems designers, and middle management people have come to think of systems for formatted file management and report generation as coextensive with management information systems, even though such systems are a travesty on what management wants and needs.

If Whorf's hypothesis is true, it can be turned around and applied in reverse. We, ourselves, build the artificial languages for programming computers. As language builders, we will have significant effects on those who use them. Thus, when we apply this Whorfian notion to artificial languages we ourselves can build, a new perspective opens up. The computer is a uniquely powerful instrument for conceptual design and structuring. The "special application" languages we provide are the major interface between the "designer" and this powerful design laboratory. The conceptual elements and syntactic tools made available to him cannot but influence the effectiveness, even the direction, his work will take.

Who is this "designer" in the case of the computer? He is the industrial manager who is seeking to direct the intricate interactions of product lines, production capabilities, markets and finance. He is the anthropologist searching for those causal laws of human behavior that lurk within his voluminous data on family relationships, vocational histories and patterns of ownership. He is the test engineer whose task is to chart the strengths and weaknesses of an aircraft wing from the thousands of data points of a destructive test procedure. He is the government agency head whose response to public need is dependent on his sensitivity to changing conditions as reflected in census statistics and spot surveys. Can we indeed say that all of these are well served by FORTRAN or PL/I? Is it not obvious that the basic conceptual elements of each of these diverse areas of computer application are themselves equally diverse, each with their own logical interrelationships and implicit assumptions? This logic and these assumptions can be built once and for all into the software support of a special application language. They can exist at that tacit level which is natural for the given application area and free the creative mind of the manager, researcher or engineer to build his programs on a conceptual platform appropriate and natural for his concerns.

We would like to illustrate this point by a reference to experience in the area of theorem proving on the computer. The resolution strategies of Robinson[2] and their refinement by a number of researchers has progressed to the point where interesting applications can be made to highly restricted areas such as the control of robots that move blocks.[3,15] However, application of theorem proving to problems of everyday business appears to be a long way away. One of the reasons for this is the necessity, if theorem proving is to give realistic results, to describe in specific axioms the very large number of tacit assumptions that are implicitly understood in these areas. Using explicit statements of these assumptions turns out to be very much more expensive in processing time than building them implicitly into heuristics and procedures. Think of the task of stating all of the tacit assumptions that underly just the personnel files of a modern business. But these same assumptions, known implicitly to the applications programmer, are built into the procedures for processing that data. To be sure, these procedures are specific to the application area. Indeed it is just such idiosyncratic procedures that become embodied as the interpretive routines of a language which is "natural" to such a specialized area. An example is the inclusion of efficient differential equation solving algorithms in applied mathematics systems such as NAPSS,[4] algorithms that are automatically invoked in response to natural syntax for the expression of such an equation. From there on, they are no longer conscious considerations in further program writing, or processed internally as axioms. They are now implicit, pressed down below the level of the explicit considerations of the researcher or manager who is using his own natural language. There are two advantages here. One is the advantage that accrues to the user in the efficiency of his language in dealing with matters that are directly of his concern. The second is the advantage in the underlying processing algorithms that make use of the underlying implicit assumptions of the domain. *These enormous economic advantages that the computer can thereby put at the fingertips of the specialist in his area are the true source of the pressure for special application languages.*

THE PRESENT RESOURCE ENVIRONMENT

When the computer was the scarce resource and programming was the domain of a small community of professionals—and indeed these conditions still residually apply—the powerful multipurpose language was the natural tool for man/computer communication. But this situation has been changing. We will examine four significant developments in this regard.

Training of computer scientists

In the first place, our universities are turning out a swelling stream of graduates in each of the various professions who, over and above their professional specialty, are also knowledgeable in computing. Our major schools, and even many smaller institutions, include as an integral part of their professional curriculum creditable courses in programming and computer systems. So far, most of these are limited to a few widely used algebraic languages such as FORTRAN. However, more and more, specialized languages are being taught: COBOL in business schools, LISP in psychology, SIMSCRIPT[5] in industrial management, various applied mathematics languages, such as NAPSS,[4] in areas of engineering, data analysis languages, such as SPSS,[6] in the social sciences.

More important than the large number of students taking these computing courses is that much smaller minority in the various disciplines who take the more advanced courses in the computer science curriculum. They go into their professional areas fully competent to bring to their disciplines the full power of the computer, and they are highly motivated to do just that. It is these cross disciplinary people who will spawn the new languages for their specialized application areas. They will know the conceptual elements and implicit logic essential to their substantive discipline and they will also know how to embody these within efficient programs and with linguistic sophistication reflecting their training in computer science.

Let's review a case history in this regard. Dr. Harry Markowitz received his doctor's degree in economics, concentrating in areas which required a high degree of knowledge of mathematics and computing. At the RAND Corporation his work involved the development of some economic models and their implementation on the computer. He then became one of the leading scientists in the RAND logistics project where he was instrumental in the development and application of simulation techniques. Subsequently Markowitz worked within the executive office of the General Electric Company, applying digital simulation to manufacturing and corporate problems. As he developed his simulation programs, his style and program organization became clearer and more modularized. Also, the scope of application of his work grew. Upon returning to the RAND Corporation, it was natural for him to distill from all of the various simulation programs he had written and supervised, a general technique that was widely applicable. The result was one of the first and still one of the best discrete simulation languages, SIMSCRIPT.[5,20]

The number of such able, interdisciplinary people need not be large to make a significant impact, and both the number and quality are growing.

Advances in hardware

The second development we would like to cite is in the hardware area. The price of computers and peripheral equipment continues to come down. There will, of course, always be the super-jobs which can utilize unlimited amounts of computer time. However, as computer costs drop, not only is the market widened, but the experienced user can afford to shift more of his task onto the computer and thus demand that the computer do more for him. One form of this extra service is to move the man/machine interface closer to the man. One of the principal ways this can be accomplished is through the development of languages that are more natural for the various application areas.

However, more significant than the general downward trend of cost per cycle time is the advent of the mini-computer. Many project teams in industry, in the university and other research institutions, in all walks of life are now finding they can afford their own computer. Up to now the main market for mini-computers has been in areas of process control or in connection with special input devices and sensing equipment. Those applications of computers which are not tied directly to special sensors or control devices but which require complex application programs have tended to gravitate to central computing centers with their large programming staffs and facilities for batch processing the many debugging runs that characterize application software development. But the cost advantage of one's own mini-computer which avoids the growing overhead costs of the big computer centers will tend to reverse this. A preliminary look indicates that very high level conversational languages can be implemented as dedicated mini-computer systems.

The development of effective programming languages for specialized application areas can be instrumental in the opening up of sizable markets for the mini-computers. As these versatile devices are brought to bear on the sophisticated problems of competent professionals, with their small group of captive programmers, we can again and again expect to see the evolution from special application programs, to module libraries, to monitors, to special application languages. Once such languages are developed, they stimulate similar installations elsewhere. The heavy attendance at specialized area sessions at computer conferences attest to the interest in learning about just such developments. Because of the low cost and considerable capability of the mini-computer, one can expect a general shift toward single user installations with a corresponding increase in independence and innovation. The result cannot help but have a considerable effect on the development of languages for specialized application areas.

Developments in systems programming

The third general area of development is that of systems programming of which extensible programming language research is a part. Here the rapid growth of interest and importance of extensible programming languages is a key development that will have a profound effect on the proliferation of specialized application languages.[21] To see this, one needs to be aware of the nature of the problems that those working in the area of extensible languages are attacking. The deep problems of this domain have to do with the building of optimizing compilers. It is no great trick for the experienced systems programmer to build a programming language that provides for complex structures declarations, at least when he can implement these without consideration of run time efficiency or storage management. The real problems are how to achieve optimization of both storage and computing efficiency. Important inroads into this difficult and central area are being made. We cite, for example, the work of Cheatham and his group at Harvard.[7]

As work progresses on extensible programming languages, one of its primary applications will surely be to the definition of higher and higher level languages, languages that include more and more implicit structural logic, indeed languages that fit closely the specialized needs of significant application areas. The ability to produce such languages efficiently and at the same time to retain reasonable levels of optimization in the actual encoding provides a powerful tool for specialized language building. These specialized application languages need not be limited to domains outside computer science. The language Planner, developed by the Artificial Intelligence Group at MIT,[8] is an excellent example of such a language to be used by computer scientists themselves. In this case it is built on LISP which is surely an example of an extensible language.

Science of linguistics

The fourth area of development is linguistics itself. We are indeed learning more and more about the structures of languages and the underlying reasons why language is such a powerful tool for conceptual structuring and communication. In particular, we are rapidly gaining sufficient knowledge of the mechanisms of natural language so that useful segments of natural language can be understood and responded to by the computer. Our own work on the REL system is a good example of where this is being successfully accomplished.[9,10] In our system, we combine results from recent work in theoretical linguistics, namely modern deep case grammar as developed by Fillmore,[11] with an efficient application of the notions of syntax-directed compiling. We believe we are only a very few years away from an English driven system for conversational data analysis with levels of efficiency that will markedly improve upon present formal language batch systems. Another system, that of Terry Winograd at MIT has demonstrated the ability to control intelligently the complex behavior of a robot in response to directions given in a comprehensive subset of natural English.

The importance of these types of systems does not lie in some magical quality of "English." There is a rather general understanding by computational linguists working on actual systems that "natural" languages are full of idiosyncratic notions and expressions that derive from the particular domain of discourse for which they are adapted. What is important about this natural language research is not the vocabulary, which is surely not a universal of the native fluent speaker. Rather the important insights from this research concern syntactic mechanisms and their interaction with semantics. We have referred to the need in a language not only for the right conceptual elements but for a sufficiently powerful syntax that will allow the efficient expression of interrelationships, as indeed is found in natural language.

Natural language has a variety of powerful mechanisms for combining conceptual elements. Ambiguity is an excellent example. Usually, when ambiguity in a language is mentioned, one thinks of ambiguous sentences. However, consider how ambiguity is involved in phrases, i.e., segments of sentences. In the phrase:

"the computer just moved into the laboratory"

the words "computer," "moved," and "laboratory" are essentially all ambiguous when standing alone in that they do not designate a unique piece of equipment, action or location. The phrase "moved into the laboratory" also could be used in many contexts where it would have quite different meanings. However, the above phrase, when taken as a whole and in context is not ambiguous at all. This ability to use general nouns and verbs in such a way as to balance off their ambiguous referents to achieve in a parsimonious way a totally unambiguous description is a powerful and ubiquitous tool of natural language, and one that is not difficult at all to include in computer languages where the context is delimited to a specialized application area. Thus, work on natural language will indeed go far in providing both understanding and specific techniques for building specialized application languages with sufficient expressiveness to truly serve as effective design laboratories for conceptual structuring.

Our growing knowledge of computational linguistics goes beyond knowledge of natural language to all areas of language communication. In particular in the domain of graphic languages and man/machine communications using graphic consoles, specialized man/machine communication languages have been and surely will increasingly be developed.

The rapid growth in linguistic knowledge will stimulate advances beyond the immediate effect of more effective computer languages and language systems. When a domain of human knowledge expands as linguistic knowledge is expanding today, it has far reaching consequences. The simulation of more effective systems built upon these new linguistic insights, insights that span from English syntax to psycholinguistics and the mechanisms of language acquisition, cannot help but be great. And greater understanding has always led to higher degrees of specialization, specialization that spells efficiency.

TRENDS IN PROGRAMMING LANGUAGES

The pressures and developments discussed in the previous two sections give evidence that programming languages for specialized application areas will continue to proliferate. Moreover, they imply several more specific trends. We shall identify and examine several of these.

As knowledge of linguistics grows and the application of computers to specialized areas continues, we realize that our old algebraic languages—FORTRAN, ALGOL, PL/I—are really quite special purpose after all. Certainly there is a need for good algebraic languages in physical science and engineering. However, other language developments, for example LISP, have already demonstrated the need for programming languages with quite different characteristics.

The attempt to include in PL/I a wide range of data structure declaration capabilities has led to an interesting and perhaps significant development. PL/I has not replaced FORTRAN for writing special application programs. It does appear that it is being used by some systems programmers and by programmers who are writing languages for special application areas. The problem with using PL/I for these systems programming purposes is that the resulting code is not sufficiently optimized. But it is exactly in this area of optimizing compilers that progress of the greatest importance is being sought in extensible language research. The confluence of these developments will lead to a movement toward powerful system programming languages of the extensible type. They will be based upon our increasing understanding of the nature and importance of data structure.

The advent of these systems programming languages will mean that we will have greatly increased ability to create languages that are carefully tailored to the conceptual and structural needs of specialized areas. The mini-computer, the computer utility and the general lowering of computing costs are creating a ready market for such language developments.

Thus we foresee that systems programmers will be turning away from their traditional preoccupation with the architecture of, and compilers for, classical algebraic programming languages. System programming will turn more of its attention to the efficient implementation of a much wider class of language mechanisms and the building of these mechanisms into a far more diverse family of application languages. These developments will obviously be greatly influenced by the spreading acceptance of conversational, interactive computing.

A second major trend will be toward natural language systems. We emphasize that by natural language we do not mean the general English of some mythical native fluent speaker. By natural we mean computer languages which reflect the conceptual elements and the syntax of the application areas in which they are employed. These languages will be specialized languages, idiosyncratic, reflecting the physical as well as the conceptual environment in which they are to be employed. For example, they will be influenced by the kinds of terminals or displays to be used, whether they refer to very large files of data, whether the application area naturally employs advanced mathematics, the interactive reaction times required, etc.

There are two technical characteristics, however, which many of these languages for specialized application areas will share. First they will tend more and more in the years ahead to be problem definition languages rather than procedural languages. The distinctions between problem definition languages and procedural languages is extensively discussed in the computer science literature without definitive conclusions.[12] Thus it will do no harm if we add to that discussion a distinction that we feel to be an important one. A procedural language talks essentially about the computer; statements in such a language are instructions to the computer about what it is to do. A problem definition language talks essentially about the domain of application; statements in such a language either describe, ask for descriptions or direct the computer concerning the subject matter of the application. It appears to us that higher level languages and languages for more highly specialized languages tend to be closer to problem definition than to procedural languages. We feel that there will be an increasing trend toward problem definition languages.

The second technical characteristic that we foresee is a trend toward natural language syntax. English as a programming language has been discussed for a good number of years and often fervently wished for but thought of as something for the distant future.[13,24,22] Through these same years solid progress has been made—in theoretical understanding of linguistic structure, in computational linguistic practice, and toward operational English language systems.[9,10,15,16,17] Pragmatic work on machine translation has been showing practical results,[18] contrary to some expressed opinions. As a result, the next few years will see operational systems for specialized areas where the structure of the language is closely related to our native English. Once this capability has been conclusively demonstrated, prejudice against it will be forgotten and we will make use of the powerful syntactic mechanisms of natural language in many application areas.

SUMMARY

Return for a final look at the Whorfian hypothesis, that language shapes the thought and culture of those who use it. As we develop powerful systems programming languages for application language development, as we incorporate the power and expressibility of natural language syntax into our application languages, as the economic costs of hardware and language software come down, and particularly as our rapidly expanding knowledge of linguistics continues to grow, a new dimension in artificial language development will come into being. We will recognize that language design is a powerful means of direction and control. The tasks of professionals can and will be directed by the languages they must use in communicating with their computers.

REFERENCES

1 B L WHORF
Language, thought and reality
MIT Press 1964 p 213

2 J S ROBINSON
The present state of mechanical theorem proving
Proc of Fourth Systems Symposium 1968

3 C GREEN B RAPHAEL
The use of theorem proving techniques in question answering systems
Proc 23rd Nat Conf of A C M 1968

4 L SYMES R ROMAN
NAPSS: Syntactic and semantic description for the numerical analysis programming language
Comp Sci Dept Purdue Univ 1969

5 H MARKOWITZ B HAUSNER H KARR
SIMSCRIPT: A simulation programming language
The RAND Corp Santa Monica 1963

6 N NIE D BENT C H HULL
SPSS: Statistical package for the social sciences
McGraw-Hill 1970

7 B WEGBREIT
The ECL programming system
Div of Eng and App Phy Harvard 1971

8 C HEWITT
PLANNER: A language for proving theorems in robots
Proc of Internat Joint Conf on Arti Intell 1969

9 F THOMPSON P LOCKEMAN B DOSTERT R DEVERILL
REL: A rapidly extensible language system
Proc 24th Nat Conf of A C M 1969

10 B H DOSTERT F B THOMPSON
The syntax of REL English
REL Project Calif Inst of Technology 1971

11 C J FILLMORE
The case for case
Universals in Linguistic Theory ed E Bach and R Harms
Holt Rinehart and Winston 1968

12 J E SAMMET
Programming languages: History and fundamentals
Prentice-Hall 1969 p 20-22 p 726

13 J E SAMMET
The use of English as a programming language
Comm of A C M 9 pp 228-230 1966

14 F B THOMPSON
English for the Computer
Proc FJCC pp 349-356 1966

15 T WINOGRAD
Procedures as a representation for data in a computer program for understanding natural language
Project MAC MIT 1971

16 C KELLOGG J BURGER T DILLER D FOGT
The converse natural language data management system
Proc of Sym on Info Stor and Retrieval Univ of Maryland 1971

17 W A WOODS
Transition network grammars for natural language analysis
Comm of A C M 13 pp 591-606 1970

18 S PERSCHKE
Machine translation, the second phase of development
Endeavour 27 pp 97-100 1968

19 J SAMMET
An overview of programming languages for specialized application areas
To be published in Proc SJCC 1971

20 *SIMSCRIPT II.5 handbook*
California Analysis Centers Santa Monica 1971

21 Proc of A C M Sigplan Conference on Extensible Languages Sigplan Notices 6 #12 December 1971

22 M D HALPERN
Foundations of the case for natural-language programming
Proc FJCC 29 1966

AMBUSH—A case history in language design

by STEPHEN WARSHALL

Applied Data Research, Inc.
Wakefield, Massachusetts

INTRODUCTION

AMBUSH is a language in which the user can describe a materials-processing/transportation network in an allegedly readable form. The AMBUSH Compiler transforms this description into a set of linear equations and inequalities and edits this set into an input file for a linear programming package.

Although the language is powerful enough to handle both materials-processing and transportation, its design is heavily biased toward processing, with transportation expected to appear only occasionally, around the "edges" of the network. Thus, a typical subject for AMBUSH description might be a very complex oil refinery with a few arcs representing transport of crude oils into and final products out of the refinery.

AMBUSH was designed by Applied Data Research for the Shell Oil Company, to facilitate the preparation of very large linear programming problems. This project followed a long history of experimental language design by Shell and its contractors, which had yielded a very valuable crop: Shell was able to summarize, fairly confidently and economically, the set of meanings to be expressed. The present language is the result of collusion (and sometimes collision) between Shell's body of experience and ADR's background in language and compiler design.

AMBUSH, as actually designed and implemented, represents a conscious compromise between two styles of language design: the "template" style, with a large set of different statement types, corresponding roughly one-to-one to the kinds of utterances the user is accustomed to; and the recursive style, in which the designer hunts for a minimal number of logical primitives sufficient to express all meanings and devises the shortest (in number of types) and most highly recursive grammar possible for combining the primitives. This stylistic tension and, indeed, the historical sequence from pure template languages toward more recursive ones exhibited in the AMBUSH project are, we believe, common characteristics in the development of languages for new problem areas.

Whenever any body of informal discourse about a subject is analyzed in order to create a formal language for saying the same things, the primary effort is concentrated on verifying that the formal language under creation is complete—that one can express every meaning—with little concern for ease of use or its close relative, grammatical simplicity. The first requirement is met by letting the language arise from, and comparing it to, a large body of utterances in the informal discourse. The loom of this body of utterances generally causes the second criterion—essentially that of linguistic "goodness"—to be quickly reinterpreted to mean superficial similarity to the informal language. Forms in the slowly developing formal language are adjudged good insofar as they "look like" the corresponding forms in the informal one. This perfectly natural criterion generally leads to first-cut designs which, although very valuable (for they effect the desired formalization), usually exhibit certain characteristic properties:

a. A syntax which is "broad" rather than "deep." By this we mean that they tend to have a large variety of different, special syntactic forms—each aping a familiar English one—rather than a small number of forms and a small set of rules for recursively combining them.
b. A proliferation of reserved identifiers, each of which once again apes a commonly used word in the informal language, even when several of these identifiers may be logically identical in function.
c. Extremely rigid format requirements, which mechanically copy the forms in which informal information is currently expressed.

These properties tend to characterize languages which, however easy to read, are very difficult to learn to write in, since each type of utterance has its own special construction rules and its own reserved identifiers; difficult to compile, since the grammars are so broad; and difficult to extend, since the grammars are so non-uniform and format-controlled.

It has been our general experience that a first-cut language design should be viewed as a formal characterization of the class of required utterances, rather than as an end product. What is required next, of course, is a redesign embodying deliberate infusion of the recursive style. The ever-present danger is the obvious one of going too far: users are willing to change their habits only so much, and a language which demands a complete reorganization of their styles of thought and expression sets an unacceptably high price, for perfectly understandable reasons.

In this paper we do eventually present a skeletal grammar of the AMBUSH language, as it was actually implemented. Our primary purpose, however, is not to describe a language. Rather we intend to treat the AMBUSH project as a case history of the language design process, which may give substance to our general remarks about the shift from template to recursive style.

We begin by summarizing how an engineer thinks about his model of an oil refinery: the appearance of his graph, the kinds of assertions he makes about flow in the graph, the packages of information he thinks of as single elements of model description. We then describe, in general terms, the structure of languages in the template style for expressing the engineer's meanings. Next we reexamine the engineer's utterances from the point of view of a designer in the recursive style, concerned with logical minimality and searching for opportunities to shorten the grammar by the use of recursion. In this last section, the general outline of the final grammar of AMBUSH takes shape. Then we present that grammar, so that the reader can verify our claim that AMBUSH is indeed a compromise: a language basically in the recursive style, but with elements of the template style still remaining, to render it more familiar-looking, and thus more palatable, to the user.

HOW THE ENGINEER DESCRIBES HIS MODEL

The directed graph

An engineer conceives of his model as a directed graph (with no self-loops), with labels on the nodes and arcs. [The bracketed examples in this paragraph refer to the figure in Appendix I.] The nodes of the graph are labeled with the names of the processing units of a plant or refinery, or the staging points of a transportation network ['EGG.PLANT,' 'BLEND.UNEX,' 'DEPOT']. The arcs of the graph represent the paths of flow of the various materials in the model. Each arc is a path for exactly one material and is labeled with the name of that material ['Exotic Liquid,' 'Unexciting Prod.']; if several materials flow from one node to another, there will be several "parallel" arcs (arcs with a common head node and a common tail node) between the nodes [Universal Solvent and Residue both flow from ICE.PLANT to BLEND.UNEX]. Node names are unique (no two nodes bear the same name), while arc names are not (several arcs may bear the same name: the same material may flow in numerous arcs of the graph [Residue flows from ICE.PLANT to BLEND.UNEX, to TRUCK, and to DEPOT1, and from DEPOT1 to DEPOT2]). Parallel arcs *must* have different labels: that is, there can be but one arc carrying a given material from a given node to a given other node.

Thus, an arc of the graph is completely specified by an ordered triple of names: ⟨node name, arc name, node name⟩.

The variables of the model correspond to the arcs of the graph or, more precisely, to the flow along the arcs. That is, the engineer states his restrictions on the model and his costs in terms of flows on arcs, and seeks an optimal solution stated in terms of flows on arcs.

The arcs of the graph will become the columns of a conventional LP matrix; the restrictions on the model will become rows of the matrix; and the various costs incurred in the model will become the objective function(s) of the matrix.

Restrictions on flow in the graph

Material balance

In general, at every node of the graph which possesses both inputs and outputs, the engineer wishes to relate the various flows entering to the flows leaving the node. These relations are conventionally expressed, in the LP matrix, as a set of "material balance equations" (rows). Each equation expresses the total amount of a single material flowing out of a node as a linear expression in the flows of the materials entering the node. The coefficients of such an expression are called "yield factors," and the engineer thinks in terms of these yield factors. It is important to note that a single material balance equation, while a natural element (a single row) of the LP matrix, does not necessarily correspond to a natural single utterance for the engineer. He may, for example, refer to a handbook of standard processing units which contains an array (inputs vs.

outputs) of yield factors, each of whose columns corresponds to a single equation; or an array of outputs vs. inputs, with rows corresponding to equations; here the entire array corresponds to a single utterance for the engineer. On the other hand, the engineer may assemble his description of material balance at a node from several sources, each of which supplies a few yield factors—each factor associated with an input/output pair—in no particular order. In sum, the engineer's natural utterance is an arbitrary subset of the yield factors at a single node, or a complete array of these yield factors organized in either of two ways.

Quantity restrictions

The engineer may set a limit (maximum, minimum, or fixed amount) on the total flow through a set of arcs. Each such restriction becomes a row of the LP matrix. It is noteworthy that the set of arcs in a single restriction is not at all arbitrarily chosen, but is always either a subset of the arcs entering a single node or a subset of those leaving a single node. This limitation imposes no logical restriction on the engineer, for he can always introduce further arcs and nodes so as to create a set of input arcs to a single node (or even a single arc) whose flow will be precisely equal to the sum of the flows in an arbitrary set of arcs. The point here is not that a certain kind of arc set is logically sufficient, but rather that it is the only kind that the engineer in fact uses as a domain for quantity restrictions.

"Quality" restrictions

The engineer may set a limit on the value of some physical property for the mixture of the flows on several arcs. For example, three arcs might carry three different materials each with a certain specific gravity. The engineer wishes to restrict the relative flows on the three arcs so that the specific gravity of the total flow satisfies some limit (maximum, minimum, or fixed value). If we denote the flows in the different arcs by f_i and the respective specific gravities by g_i, the engineer is setting a limit on the value of $\sum_i f_i g_i / \sum_i f_i$. Once again, as with the quantity restrictions, the set of arcs is always either a subset of the arcs entering, or a subset of the arcs leaving, a single node.

Ratio restrictions

The engineer may set a limit on the ratio of the total flow in one set of arcs to the total flow in another set of arcs. Again, it is a feature of user behavior that the union of the two sets of arcs referenced in a single ratio restriction is always a subset of the arcs entering or a subset of the arcs leaving a single node.

Cost and income

The engineer may wish to state that flow along certain arcs incurs cost or generates income. Eventually, all his remarks of this kind will become the objective function of the LP problem, a linear expression in flows on arcs which is to be minimized. The engineer's typical utterance does *not* correspond to an entire objective function, but rather to the supply of a single coefficient value for the objective function; such a coefficient value may apply to one arc or to a set of arcs: thus, the engineer may say, in effect, "flow on any of *this* set of arcs costs $3.50 per unit flow." It is no serious distortion of user behavior to say that the set of arcs to which a single cost/income coefficient applies is always either a subset of the arcs entering, or a subset of the arcs leaving, a single node.

TEMPLATE-STYLE LANGUAGES

Even from the rather brief picture given above of the engineer's view of his model, it should be clear that he has a fairly large number of different—or what he believes are different—kinds of things to say. He must be able to describe a graph, supply a set of yield factors, supply an array of yield factors in input/output form, supply the array in output/input form, state quantity restrictions on the output side of a node, or the input side, assign cost to an arc, assign income to an arc, and so forth.

It is important to note that, although the difference between some two elements of this list may seem quite trivial to the reader (input/output form versus output/input form of an array of yield factors, for example), the difference may look enormous to the engineer, who obtains the two kinds of information from different sources, uses them in different modeling contexts, and discusses them in a different vocabulary. In our effort to summarize the engineer's utterances concisely, we have made use of logical similarities between utterances which might, to him, look quite different. Thus, in our description of quantity restrictions, we took advantage of the fact that to set a quantity maximum is not unlike setting a quantity minimum. In some real problem, however, all the maxima may come from physical limitations (pipeline capacities, say), while all the minima may reflect business obligations (requirements to buy at least so much from various suppliers). The engineer in this situation is not working with maxima and minima (which even *sound* similar) but with capacities and obligations, which he may customarily treat with quite dissimilar vocabularies.

A common characteristic of the early language designs for the description of these models was a strong tendency to preserve, more or less uncritically, the many logically empty distinctions between "different kinds" of information. Thus, such a language would typically consist of a large number of dissimilar-looking templates, composed of reserved words; each template would correspond to one of the different kinds of information, and the reserved words would guarantee a familiar-looking quasi-sentence appropriate to that kind. The holes in the templates were to be filled with nothing more complex than lists of numbers or lists of names.

It would be grossly unfair to early workers in the area to suggest that the number of templates in any real language design was anywhere near as large as completely uncritical acceptance of the engineer's discriminations would have implied. Considerable, and quite fruitful, effort was devoted to the reduction of this number to even more manageable limits. Nonetheless, it *is* fair to say that even the latest of the pre-AMBUSH languages retained all the stigmata of the template approach:

a. An extremely broad and shallow grammar: a very large number of syntactic types at the "statement" level, with almost no phrases in common (above the level of name list or number list).
b. A very large number of reserved words.
c. Inconsistencies of lexical style: the use of punctuation marks as separators when the templates were to look sentence-like, the use of spaces or position-on-card to delimit symbols if the templates were to look like arrays.
d. Prefix characters to signal statement type: since the templates possessed little or no common phrase structure, the analyzer was likely to consist of many essentially independent recognizers entered through a common switch and thus a prefix character was required to control the switch.

ANOTHER LOOK AT THE ENGINEER'S UTTERANCES, BY DESIGNERS IN THE RECURSIVE STYLE

Reduction in variety of user's statement forms

Let us set aside for the moment the problem of specifying the structure of the graph itself, and confine our attention to those of the engineer's utterances which include numerical information—these are the statements of restrictions or remarks about cost and income. The most striking fact about these numerical assertions is that each concerns an extremely small portion of the whole graph. Thus, a remark about material balance always talks about a subset of the arcs entering or leaving a single node; and the other numerical assertions only refer either to a subset of the arcs entering, or a subset of the arcs leaving, a single node.

This immediately suggested the possibility of turning the language "inside out" in the following sense: instead of having one statement type per "kind of information," each containing its subgraph specification after its own fashion, we could have a far smaller number of statement types, one per kind of subgraph, each containing the kind-of-information specification in a more uniform fashion. This inversion had several attractions:

a. It dropped to a lower syntactic level the vast array of special forms, with their reserved words and need for special recognizers; above this level, phrase structure could be quite simple and uniform.
b. It simplified the job of persuading the user of the logical emptiness of many of his discriminations; if the template for pipeline capacities looks completely different from that for purchase obligations, with reserved words scattered non-analogously throughout the two forms, it is hard to convince him that they are logically similar; if, however, he sees the differences entirely embodied—reserved words and all—in two short phrases which fit in exactly the same larger structure, he becomes far more willing to drop the reserved words for his 10 different kinds of quantity restriction and simply say "maximum" or "minimum."
c. It became trivial (indeed, almost inevitable) in grammar design, and very palatable to the engineer, to take advantage of the symmetry between the inputs of a node and its outputs. The syntactic type for discussing the one could look identical to that for discussing the other, to within a reserved word or two.
d. It suggested some interesting possibilities of nested information. The engineer frequently had several things to say about the same subgraph, or about two subgraphs, one of which was a proper subgraph of the other. We could perhaps arrange our grammar to permit him to nest his subgraph specifications, and to associate with each nesting level a *set* of (say) restrictions.

The redundancy of graph specification

A notable feature of the numerical assertions described above (the restrictions and income/cost utterances) is that, whether they be written in engineer's jargon, some template language, or in some new, recursive language, they inevitably contain information about the structure of the graph. This is perhaps obvious, for—as we have earlier indicated—every numerical assertion must contain a specification of the subgraph to which the assertion applies.

Thus, to say that the maximum amount of M flowing out of A is less than five surely suggests that the graph contains a node named A with at least one arc labeled M leaving it. Every numerical assertion contains some such topological information, and the set of such information collected from all assertions appeared to yield a fairly good picture of the overall shape of the graph. Initially, this redundancy was expected to provide a check: the topological implications of each assertion might be verified against a previously given graph description.

Then a further observation was made. A very large number of the nodes in a graph did not represent either processing units or staging points, but were essentially formal: they represented "pools" of intermediate products. Within a model, the engineer deals with two kinds of materials: the "pooled" ones, like intermediate products in a refinery, such that any consuming node can obtain them from any producing node; and the non-"pooled" ones, like raw materials and final products, which flow from each producing node to certain consuming nodes, and no others. For each pooled material, the engineer establishes a single pool represented by a node, with arcs to the node for each producer and arcs from the node for each consumer.

Moreover, since our user population modeled very large refineries with only a small amount of transportation at the edges of the graph, it turned out that virtually all materials in a typical model were of pooled type. This implied that, if we required the user to declare explicitly his non-pooled materials (which should be but a small burden, for they were not numerous), the Compiler could draw far stronger topological inferences from his numerical assertions than we had originally thought. Thus, from the example given earlier in this section, one can deduce not only that at least one arc carries M out of A, but—if M is pooled—that the arc must go to the pool for M and must be unique (since parallel arcs must have different labels).

This analysis led to the conclusion that the set of topological implications contained in all the engineer's numerical utterances amounted to a virtually complete description of the graph, and thus separate statement types designed for graph specification were quite unnecessary. True, there might be an occasional piece of graph left ill-defined after the numerical statements were written, but our statements for making numerical assertions were going to incorporate the specification of little subgraphs to which a set of numerical remarks would apply. If we simply permitted these types to contain an empty set of numerical remarks, we would have a mechanism for describing little subgraphs, thereby filling in the holes in the graph definition. (The fact that this degenerate form would be inappropriate for describing a whole graph was entirely irrelevant.)

THE GRAMMAR OF AMBUSH

The grammar of AMBUSH derives directly from the observations of the preceding section. There is no statement type intended exclusively for graph description; there are three statement types for making numerical assertions, differentiated semantically by the kinds of arc set (subgraph) they discuss:

a. The YIELDS statement, for discussing the inputs and outputs of a node; this is used exclusively for the supply of yield factors.
b. The SENDS statement, for discussing subsets of the arcs leaving a node; this is used for all numerical assertions except the supply of yield factors.
c. The TAKES statement, for discussing subsets of the arcs entering a node.

(Note: In this section, we have permitted ourselves the liberty, in stating our rules of grammar, of using, without definition, the types ⟨identifier⟩ and ⟨numex⟩, which latter type corresponds to "numeric expression.")

The YIELDS statement

Grammar

```
              ⟨row⟩::= ⟨identifier⟩, ⟨numex⟩ | ⟨row⟩,
                       ⟨numex⟩
         ⟨row list⟩::= ⟨row⟩ | ⟨row list⟩, ⟨row⟩
        ⟨name list⟩::= ⟨identifier⟩ | ⟨name list⟩,
                       ⟨identifier⟩
⟨yields statement⟩::= ⟨identifier⟩ RUNNING
                       ⟨name list⟩ YIELDS
                       ⟨row list⟩ | ⟨identifier⟩
                       YIELDS ⟨name list⟩
                       RUNNING ⟨row list⟩
```

Examples

```
a. U YIELDS          S1,  S2,  S3
   RUNNING      S4,  .1,  .4,  .5,
                S5,  .6,  .1,  .1
b. U RUNNING         S4,  S5,
   YIELDS       S1,  .1,  .6,
                S2,  .4,  .1,
                S3,  .5,  .1
c. U RUNNING    S1 YIELDS S2, .4, S3, .5
```

Discussion

It should be clear from the first two examples (which are identical in meaning) that the grammar has been designed to facilitate supply of a large array of yield factors in either input/output or output/input form. The more "linear" style, exhibited in Example c, for the supply of a few yield factors, is identical grammatically to the array style.

It should be noted that the grammar demands that the set of yield factors supplied in a single statement always correspond to some array (the cross product of a set of inputs and a set of outputs), however small. This requirement is made less troublesome than might appear, for the Compiler systematically fails to distinguish between a yield factor of zero and no yield factor at all. Thus, the engineer can, for example, write one statement to cover all but one of the yield factors at a node (representing the missing one by zero) and a second statement to supply the omission.

The SENDS statement

Grammar

a. Modifier

```
            ⟨limit⟩::=MAX | MIN | FIX
           ⟨qlimit⟩::=QMAX | QMIN | QFIX
             ⟨cost⟩::=COST | COST0 | COST1 |
                       . . . | COST9 |
                       INCOME | INCOME0 |
                       INCOME1 | . . . |
                       INCOME9
  ⟨quantity phrase⟩::= ⟨limit⟩ ⟨numex⟩
   ⟨quality phrase⟩::= ⟨identifier⟩ ⟨qlimit⟩
                       ⟨numex⟩
     ⟨ratio phrase⟩::= ⟨limit⟩ ⟨numex⟩
                       ⟨stream/node factor⟩
      ⟨cost phrase⟩::= ⟨cost⟩ ⟨numex⟩
         ⟨modifier⟩::= ⟨quantity phrase⟩ |
                       ⟨cost phrase⟩ |
                       ⟨quality phrase⟩ |
                       ⟨ratio phrase⟩
```

b. SENDS Statement

```
    ⟨stream/node factor⟩::= ⟨identifier⟩ |
                            (⟨stream/node
                            expression⟩)
      ⟨stream/node term⟩::= ⟨stream/node
                            factor⟩ |
                            ⟨stream/node
                            term⟩,
                            ⟨modifier⟩
⟨stream/node expression⟩::= ⟨stream/node
                            term⟩ |
                            ⟨stream/node
                            expression⟩,
                            ⟨stream/node
                            term⟩
          ⟨sends clause⟩::=SENDS
                            ⟨stream/node
                            expression⟩ TO
                            ⟨stream/node
                            expression⟩ |
                            SENDS
                            ⟨stream/node
                            expression⟩
       ⟨sends statement⟩::= ⟨identifier⟩
                            ⟨sends clause⟩
```

Examples

a. Modifier

1. MAX 50
2. SPEC.GRAV QFIX .75
3. MAX .5 (M_1, M_2)
4. COST3 −3.50
5. INCOME3 3.50

b. SENDS Statement

1. U SENDS A, MAX 50
   The maximum quantity of A output by U is 50.
2. U SENDS (A, MAX 50, MIN 40, B, MAX 50), COST 3.50
   The maximum quantity of A output by U is 50.
   The minimum quantity of A output by U is 40.
   The maximum quantity of B output by U is 50.
   A cost of 3.50 is incurred per unit of either A or B flowing out of U.
3. U SENDS (A, MAX 50, MIN 40, B, MAX 50), COST 3.50 TO X, MAX 10, Y, MIN 10
   All the meanings included in Example 2, plus:

The total flow of A or B from U to X has a maximum of 10.

The total flow of A or B from U to Y has a minimum of 10.

4. U SENDS A, B TO X

   This example includes *no* numerical assertion, and is thus "degenerate." It informs the Compiler that there is one arc carrying A, and another carrying B, from node U to node X.

5. U SENDS A, MAX .5 (B, C, MIN 10)

   The minimum quantity of C output by U is 10.

   The quantity of A output by U is nc more than .5 times the total quantity of B and C output by U.

Discussion

The four varieties of modifier correspond to the four kinds of numerical assertion (other than supply of yield factors) covered in earlier sections: quantity, quality, and ratio restrictions and income/cost assertions. The large number of reserved words in the definition of ⟨cost⟩ derives from the desire of the user to define several objective functions and from his preference *not* to treat an income as a negative cost (modifier Examples 4 and 5 are identical in meaning).

Each modifier must apply to a certain subgraph (set of arcs) of the graph and the only difficulty in understanding the SENDS statement is that of learning the rules which determine to which subgraph a given modifier applies.

First it should be noted that every SENDS statement begins with an identifier; this names a node of the graph, which we will—in this section—call the "subject node" of the statement. The word SENDS indicates that we are discussing arcs leaving the subject node. Next follows a stream/node expression in which appear identifiers which are the names of physical materials. Within this expression, such a name M is to be read as shorthand for "the set of all arcs carrying M out of the subject node," and thus defines a subgraph.

The syntactic type ⟨stream/node factor⟩ corresponds to the notion of subgraph. If a given stream/node factor is a single identifier, we have already indicated what subgraph is meant. If the factor is a parenthesized expression, the subgraph meant is the union of the subgraphs defined by all material names found in the expression. Thus the stream/node factor (M_1, M_2) is to be read as "the set of all arcs carrying either M_1 or M_2 out of the subject node."

The reader will observe that the type ⟨stream/node term⟩ is defined as a single ⟨stream/node factor⟩ followed by an arbitrarily long list of modifiers (with appropriate separation by commas). The meaning here is that each modifier in the list is to be independently applied to the subgraph defined by the factor.

In sum, then, the recursive nature of the ⟨stream/node term⟩ definition permits application of a *set* of modifiers to a single subgraph; the recursion in ⟨stream/node factor⟩ definition (i.e., the ability to parenthesize a ⟨stream/node expression⟩ to make a new factor) permits nested subgraph definition; and, finally, the recursive structure of a ⟨stream/node expression⟩ permits assertions about overlapping or disjoint subgraphs to be included in the same statement (so long as each subgraph is a subset of the arcs having the subject node).

The engineer may continue his statement, after the stream/node expression, with the reserved word TO followed by a second stream/node expression. In this latter expression, the identifiers are the names of *nodes*; each identifier N acts as shorthand for "the set of all arcs carrying any of the materials named in the first stream/node expression from the subject node to the node N," and thus defines a subgraph. The rules for applying modifiers to factors, treating parenthesization as subgraph union, etc., are exactly as before.

The TAKES statement

Grammar

⟨takes clause⟩: : = TAKES ⟨stream/node expression⟩
FROM ⟨stream/node expression⟩ |
TAKES ⟨stream/node expression⟩

Examples and Discussion

The TAKES statement is completely analogous to the SENDS statement, both grammatically and semantically. With appropriate substitution of TAKES and FROM for SENDS and TO, the examples of the previous section will be correct. With similar substitutions in the discussion section, that section will work also.

Final notes on the grammar

The basic skeleton of AMBUSH consists of the three statement types described above and a few declarations: notably, one to list non-pooled materials and another for the supply of physical property values for materials which participate in quality restrictions.

The language has a full macro capability, including the insertion, at macro-expansion time, of actual parameters for formal parameters in the macro definition. Macros are, of course, used for abbreviation and to reduce copying errors. But they also serve as a way to parametrize a model: an identifier may be used in place of any numeric quantity, and its value may be changed from run to run. Other methods of parametrization, without recompilation, are also available.

For the sake of completeness, it should be added that:

a. The Compiler treats the AMBUSH program as a string, and consumes 72-column card images stringwise without format requirements.
b. AMBUSH (like many languages without a statement terminator) has a statement initiator (the # character).
c. An AMBUSH program is bracketed by the reserved strings BEGIN and END.
d. An AMBUSH identifier is built of alphanumerics and the period character; the initial character must be alphabetic and the character limit is twelve.
e. An AMBUSH number is a string of up to 14 digits among which at most one decimal point may appear.
f. All arithmetic is performed in floating point.
g. Numeric expressions are built in the customary way out of numbers, the four arithmetic operators, parentheses, and the special functions EXP and LOG. Unary minus is permitted. The evaluation rule is left-to-right and gives equal precedence to * and / and equal precedence to + and −.
h. A comment may be inserted between any two tokens of the language (roughly, in any place a space would be allowable). The comment is delimited by < and >; the # sign is prohibited within comments so that, when the programmer neglects to close his comment, it will be closed automatically by the initiator of the next statement.

AMBUSH AS A COMPROMISE IN STYLES

It should be clear to the reader that the grammar of AMBUSH follows fairly directly from the observations made in Section 4, and in general exemplifies the recursive style:

a. The grammar clearly reflects the search for a logically minimal description of the user's utterances.
b. The syntax is "narrow" (a small number of types at the statement) level and "deep" (highly recursive, at least within the SENDS and TAKES statements).
c. There are relatively few reserved words.
d. Lexical strategy is uniform; the input is treated as a string.

On the other hand, there still remain certain elements of template style; for example:

a. The two forms of YIELDS statement (input vs. output *and* output vs. input) are a logical redundancy.
b. The reserved words QMIN, QMAX, and QFIX could be replaced by MIN, MAX and FIX. The ⟨qlimit⟩ forms remain as a reflection of the user's habits of thought: a quantity limit is somehow *different* from a quality limit.

Finally, it should be noted that the grammar we have given above is incomplete. Most of the differences between that grammar and AMBUSH as actually implemented come from the deliberate introduction of a few totally redundant templates to reduce the language's unfamiliarity. The interested reader will find several examples of these templates in the sample problem in Appendix I.

The reader will probably have noticed a feature which makes AMBUSH rather different from most languages which programmers use: there are no imperative statements in the language. All the sentences the programmer writes are in the indicative mood: they are either declarations, giving global information about the materials which flow in the network, or are local statements about individual nodes of the network. Thus the language is not algorithmic—there is no flow of control, no modification of values in the course of computation. This implies that the order of inputting sentences to the compiler is irrelevant, and this is essentially true. (The exception is that, if conflicting numerical values are given, the last one input is used.)

STATUS

The AMBUSH Compiler was written for the IBM 360/65 and has since been moved to the 360/85; it was designed to interface with the IBM LP package (MPS and MPSX). By the Fall of 1971, it had also been moved to the UNIVAC 1108, and was being shaken down on that hardware.

The AMBUSH Compiler was delivered near the end of 1969, and has been in regular use by Shell since that time. We understand that, as of late 1971, there were approximately 35 qualified AMBUSH programmers at Shell. The language is, apparently, fairly easy to learn, since fully half of the 35 received no formal instruction: they were given a language manual and a certain amount of hand-holding by more experienced users.

The language appears to permit the expression of most meanings fairly naturally, although occasional user models have exhibited structures which were unanticipated at language design time (and thus expressible only clumsily). The frequency of the need for tricks and clumsy expressions tends to reduce with time, since the need arises less from the nature of the application than from the habits of thought of the user. Experience with a new language inevitably alters the style of thought. Clear understanding of the problem comes to *mean* clear expressibility of the problem in the programming language.

Applied Data Research has been licensed by Shell to market the AMBUSH package in the United States and Canada. Initial explorations of the suitability of the language for various classes of LP users were under way at the end of 1971.

ACKNOWLEDGMENTS

We wish to express our appreciation to the many employees of the Shell Oil Company who gave unstintingly of their time and expertise in the course of this language design. We further wish to thank Shell for the use of the sample problem in Appendix I.

The Compiler for AMBUSH was actually built by a team led by Naomi Rotenberg; the other members of this crew were Barbara Butler, Margaret Carlin, and David Goldberg.

Finally, the author wishes to acknowledge the assistance of his colleague, Kirk Sattley, who undertook overseeing the final preparation of the manuscript when the author was out of the country.

APPENDIX I: EXAMPLE PROBLEM

A simple problem in processing/transportation typical of the process industries follows. While only a small problem, it permits invoking most of the AMBUSH syntactic forms. The AMBUSH description (a listing of the input to the AMBUSH Compiler) follows the statement of the problem itself.

A few statement forms are used in this example which are not described in detail in the paper, or in the skeleton grammar. The reader will probably understand them immediately from the problem description, but in any case, some explanatory notes are given following the listing.

Statement of problem

Choose the most profitable set of operations for the following situation.

The Archetype Manufacturing Company Inc., operates two processing plants:

1. ICE processing plant
2. EGG processing plant

The ICE plant operates with an inexpensive raw material which is obtainable as follows:

SOURCE	COST $/TON	MAXIMUM AVAILABILITY, TONS/DAY
Arkansas	7.00	120
Louisiana	7.50	300

Furthermore, Archetype has a long term contract with the Louisiana supplier guaranteeing the purchase of at least 180 tons/day at the $7.50/ton price. The ICE plant can process up to 250 tons per day of raw materials at a cost of $.20 per ton yielding three products as follows:

	RAW MATERIAL SOURCE	
PRODUCT	ARKANSAS	LOUISIANA
Gas	.10	.12
Universal Solvent	.70	.74
Residue	.20	.14

The EGG plant operates with a more expensive raw material obtained as follows:

SOURCE	COST $/TON	MAXIMUM AVAILABILITY, TONS/DAY
Alaska	20.00	100
Labrador	25.00	150

The EGG plant has capacity to process all the raw material available at a cost of $.50 per ton and produces two streams.

	RAW MATERIAL SOURCE	
PRODUCT	ALASKA	LABRADOR
Exotic Liquid	.60	.70
Reject	.40	.30

The process requires .11 tons fuel gas per ton of raw material. The fuel gas must be obtained from the ICE plant. Any fuel gas not used in the EGG plant must be burned on site.

The Exotic Liquid can be blended with Universal Solvent to produce Exotic Product which has the following specification requirements.

	EXOTIC PRODUCT SPECIFICATIONS
Blue Color Index	12.0 Minimum
Specific Gravity	1.1 Maximum

The corresponding properties of the two components are:

	BLUE COLOR INDEX	SPECIFIC GRAVITY
Exotic Liquid	11.5	1.2
Universal Solvent	10.0	0.9

In order to meet the blue color index, Blue Dye must be added. One gram of Blue Dye will raise the blue color index of one ton of Exotic Product .7 units and will have an insignificant effect on specific gravity. Blue Dye costs $.80 per gram.

Universal Solvent can also be blended with Reject from EGG plant and Residue from ICE plant to make Unexciting Product. The blend has the following formula:

Universal Solvent	.25
Reject	.45
Residue	.30

All Reject must be disposed of in this manner. Any remaining Universal Solvent and/or Residue may be sold as produced.

All products leave Archetype by Pipeline except Residue which can also be removed by Truck. Pipeline carries products to Depot 1 and from there to Depot 2. Pipeline pumping limitations for each product and prices and demands at each depot are given below.

	MAX PIPELINE FLOW		SALES			
	TONS/DAY		DEPOT 1		DEPOT 2	
PRODUCT	ARCHETYPE/ DEPOT 1	DEPOT 1/ DEPOT 2	DEMAND TONS/DAY	NET PRICE $/TON	DEMAND TONS/DAY	NET PRICE $/TON
Exotic Product	500	225	125 Max.	40	120 Max.	35
Universal Solvent	400	175	60 Max.	30	20 Fix.	25
Unexciting Product	450	200	120 Max.	15	80 Fix.	10
Residue	300	125	Unlimited	3	Unlimited	3

Note that the pipeline capacity effects are cumulative. That is, 250 tons/day of Exotic Product plus 150 tons/day of residue would be permissible, but would use all of the capacity.

Residue removed by truck costs Archetype $.50/ton for hauling it away. In addition, handling problems limit to 30 tons/day the Residue disposed of as Residue.

The AMBUSH statements describing this problem follow. The AMBUSH compiler will convert these statements into a matrix which can be optimized by an LP code to give the combination of operating choices which will maximize profit by Archetype.

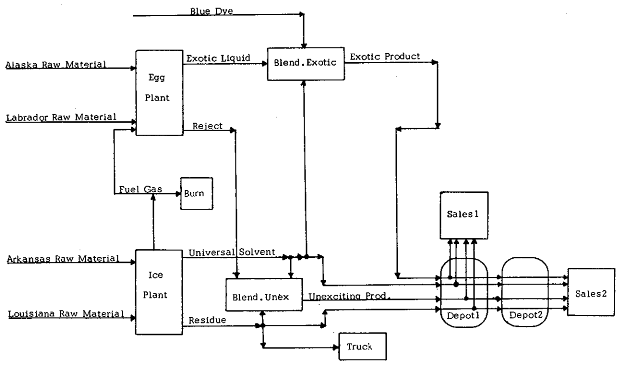


Figure 1—Archetype Manufacturing Company Process flow

```
#BEGIN                                                                       100
#MACRO:      FOUR.PRODUCT=EXOTIC.PROD , UNEXCIT.PROD ,XES.SOLVENT ,          200
                          EXCESS.RESID                                       300
#NOPOOL:     FOUR.PRODUCT                                                    400
#ADDITIVE:   BLUE.DYE                                                        500
                             < >                                             600
                    < RAW MATERIAL SLATE >                                   700
#ARKANSAS    SENDS        ARKAN.IE.R.M, COST  7.00, MAX  120                 800
#LOUISIANA   SENDS        LOUIS.IE.R.M, COST  7.50, MAX  300, MIN  180,      900
#ALASKA      SENDS        ALASK.EX.R.M, COST 20.00, MAX  100                1000
#LABRADOR    SENDS        LABRA.EX.R.M, COST 25.00, MAX  150                1100
                             < >                                            1200
#PURCHASE    SENDS        BLUE.DYE    , COST  .80                           1300
                             < >                                            1400
                    < PROCESSING UNITS  >                                   1500
#ICEPLANT   ,MAX 250, COST  .20                                             1600
                             < >                                            1700
#ICEPLANT    RUNNING      ARKAN.IE.R.M, LOUIS.IE.R.M                        1800
 YIELDS                                                                     1900
 FUEL.GAS   ,                 .10     ,     .12     ,                       2000
 UNI.SOLVENT,                 .70     ,     .74     ,                       2100
 RESIDUE    ,                 .20     ,     .14                             2200
                             < >                                            2300
#EGGPLANT   ,COST  .50                                                      2400
                             < >                                            2500
#EGGPLANT    RUNNING      ALASK.EX.R.M, LABRA.EX.R.M                        2600
 YIELDS                                                                     2700
 EXOTIC.LIQ ,                 .60     ,     .70     ,                       2800
 REJECT     ,                 .40     ,     .30                             2900
                             < >                                            3000
#EGGPLANT    TAKES        FUEL.GAS     ,FIX .11       (ALASK.EX.R.M,        3100
                                                      LABRA.EX.R.M)         3200
                             < >                                            3300
                    <   PRODUCT BLENDS   >                                  3400
#PROP:       BCINDEX, SPEC.GR                                               3500
 STREAM                                                                     3600
 EXOTIC.LIQ ,  11.5 ,   1.2,                                                3700
 UNI.SOLVENT,  10.0 ,   0.9,                                                3800
 BLUE.DYE   ,    .7  ,                                                      3900
                             < >                                            4000
#BLENDEXOT   TAKES       (EXOTIC.LIQ  , UNI.SOLVENT , BLUE.DYE    ),        4100
                          BCINDEX  QMIN   12 , SPEC.GR  QMAX   1.1          4200
                             < >                                            4300
#BLENDUNEX   MAKES        UNEXCIT.PROD                                      4400
 BLENDING                                                                   4500
UNI.SOLVENT,                ,25       ,                                     4600
 REJECT     ,               ,45       ,                                     4700
 RESIDUE    ,               ,30                                             4800
                             < >                                            4900
                    < ACCUMULATE EXCESSES>                                  5000
#POOL(RESIDUE)  SENDS     RESIDUE       TO           (RESID.EXCESS,         5100
                          TRUCKS,     , COST  .50 )  ,MAX  30               5200
                             < >                                            5300
#SOLVENT.XES TAKES UNI.SOLVENT                                              5400
#BURNING     TAKES        FUEL.GAS                                          5500
                             < >                                            5600
             < DISTRIBUTE FINISHED PRODUCTS >                               5700
#DEPOT1      TAKES        EXOTIC.PROD   FROM          BLENDEXOT   :         5800
                          UNEXCIT.PROD  FROM          BLENDUNEX   :         5900
                          XES.SOLVENT   FROM          SOLVENT.XES :         6000
                          EXCESS.RESID  FROM          RESID.EXCESS          6100
```

```
#SALES.1     TAKES        EXOTIC.PROD , INCOME 40.0  ,MAX 125.     ,         6200
                          UNEXCIT.PROD, INCOME 15.0  ,MAX 120.     ,         6300
                          XES.SOLVENT , INCOME 30.0  ,MAX  60.     ,         6400
                          EXCESS.RESID, INCOME  3.0                          6500
                                        FROM         DEPOT1                  6600
#DEPOT1      SENDS       FOUR.PRODUCT   TO           DEPOT2                  6700
#SALES.2     TAKES        EXOTIC.PROD , INCOME 35.0  ,MAX 120.     ,         6800
                          UNEXCIT.PROD, INCOME 10.0  ,FIX  80.     ,         6900
                          XES.SOLVENT , INCOME 25.0  ,FIX  20.     ,         7000
                          EXCESS.RESID, INCOME  3.0                          7100
                                        FROM         DEPOT2                  7200
                    < >                                                      7300
            < SET PIPELINE CAPACITY LIMITATIONS >                            7400
#PSEUDONODE  SENDS        UNEX.PL.CAP  ,SOLVE.PL.CAP ,RESID.PL.CAP           7500
#DEPOT1      TAKES       (UNEX.PL.CAP  ,FIX (500/450 - 1) UNEXCIT.PROD,      7600
                          SOLVE.PL.CAP ,FIX (500/400 - 1) XES.SOLVENT ,      7700
                          RESID.PL.CAP ,FIX (500/300 - 1) EXCESS.RESID,      7800
                          EXOTIC.PROD ),MAX  500                             7900
#DEPOT2      TAKES       (UNEX.PL.CAP  ,FIX (225/200 - 1) UNEXCIT.PROD,      8000
                          SOLVE.PL.CAP ,FIX (225/175 - 1) XES.SOLVENT ,      8100
                          RESID.PL.CAP ,FIX (225/125 - 1) EXCESS.RESID,      8200
                          EXOTIC.PROD ),MAX  225                             8300
#END                                                                         8400
```

Notes to the listing

Card numbers:

400,500: These illustrate the form of simple declarations—a declarative reserved word, followed by a colon, followed by a list of identifiers used as names of materials. Note that FOUR.PRODUCT is a list of four material-names by virtue of the MACRO definition on cards 200-300.

400: NOPOOL tells the compiler *not* to automatically set up a POOL node for the material(s) named in the declaration. This implies that arcs carrying these materials must have both end points specified (see cards 5800-6100 for the FOUR.PRODUCT materials). Compare this to cards 2000, 3100, and 5500 which together describe the handling of FUEL.GAS: ICEPLANT "yields" it (to its pool), and EGGPLANT and BURNING "take" it (from the pool).

500: The material BLUE.DYE is declared to be an additive. This informs the compiler that the quantity of blue dye coming into a node (card 4100) has no effect on material-balance calculations, but only on quality determinations.

1600,2400: This is a shorthand statement for applying modifiers to the total capacity or throughput of a node. Formally, such a statement is interpreted by the compiler as a degenerate form of the TAKES statement, with the modifiers applying to the set of all input arcs to the node.

3500-3900: This declaration gives, for each relevant material, the coefficient to be used in "quality" calculations, described in an earlier section.

4400-4800: This is one of the "totally redundant template statements" retained in the language because of its familiarity; the MAKES . . . BLENDING statement is a variant of the YIELDS statement, used to describe a node which produces a single material, by giving the "recipe" for its product in terms of its inputs.

The data-text system—An application language for the social sciences

by D. J. ARMOR

Harvard University
Cambridge, Massachusetts

INTRODUCTION

The enormous growth of special applications software has been the subject of much speculation and debate. While many of the developments are natural expansions into new application areas, many computer specialists argue that applications programming is a prime example of the tendency to reinvent the wheel. No area receives this accusation more than that of statistical packages or languages, particularly those developed for applications in the social and behavioral sciences. A recent study by Hugh F. Cline shows that in 130 universities which grant higher degrees in these fields, there are over 170 such packages or languages in use.[1] This is more than one per university! No one could possibly maintain that all of these systems are unique; the duplication of effort has been rampant.

Now that I have said that, I want to take the other side. While I cannot defend outright duplication, I want to present some of the reasons why many of these developments were necessary and why they represent a significant stage in the evolution of computer software. In illustrating my argument, I will draw upon examples from the Data-Text system, a data processing and statistical analysis language developed at Harvard.[2] Although other systems could be used to make the same points, Data-Text may represent the most comprehensive attempt in this field to date. Besides, since I helped to develop it, I know it better than others!

EARLY DEVELOPMENTS

One insight into the rationale behind Data-Text and other similar attempts can be gained by considering the origin of languages like FORTRAN. Why was FORTRAN developed and why did it become so popular? All modern computers have had symbolic machine or assembler languages since at least the IBM 704 generation. Why did the symbolic machine languages fail to suffice for all application software, being supplanted almost exclusively by FORTRAN for problems involving numerical computations? In retrospect the answers are obvious. First, programming in a machine language requires a good deal of detailed knowledge of the internal structure of the machine; second, the atomistic nature of the instruction set means a great deal of programming effort even for simple mathematical expressions—not to speak of the coding required for routine input-output procedures. For the engineer or scientist with a particular type of mathematical expression to evaluate, the investment of time and energy in learning and programming in machine language was costly. FORTRAN and other languages like it solved this problem for many quantitative applications in the physical sciences. FORTRAN was relatively machine independent (at least by the end of the 1950's) and could be learned very quickly by those with some background in mathematics. Most mathematical formulas could be expressed in a fairly natural way, and most of the troublesome clerical problems of I/O were handled with a minimum of complexity (at least in comparison with machine language solutions). In other words, the FORTRAN-type languages were "natural" languages for those programmers working in numerical mathematical applications. The same can be said of many other languages, such as COBOL in the business realm and SNOBOL in the field of text processing.

For many other fields, however, FORTRAN is not a "natural" language. In the social and behavioral sciences, where mathematical proficiency is not predominant, a great many statistical analyses are commonly used even though their mathematical bases are not fully comprehended by the user. Two examples are the techniques of regression analysis and factor analysis. The first requires inversion of a matrix of correlation coefficients, and the second requires extraction of eigenvalues and eigenvectors. Although a competent social scientist can understand the results of these analyses, very few actually understand the mathematical techniques which are required to derive the results. (In defense of the social scientist, I should hasten to add that a great number of physical scientists I have known are likewise in the dark.) Even in the case of less complex statistical methods, many researchers are not able to provide precise computational formulas without careful review of a statistics textbook.

The result of this is that the social scientist finds it difficult to program in a language which requires that he provide the mathematical or statistical formulas for all of his ordinary analyses. Thus languages like FORTRAN, ALGOL, BASIC, PL/I, and COBOL are not natural languages for the social scientist. I think this argument can be extended to other fields as well where standard algorithms exist for data analysis problems. In these cases, if a language is to earn the label "natural," it should embody terms or macros which call in these entire algorithms, just as FORTRAN calls in a routine in response to a function name such as SQRT or LOG. In other words, FORTRAN and similar languages are natural languages for the programmer who wants to *implement* a statistical or mathematical algorithm, but they are not natural languages for those who simply want to *apply* those algorithms to analyze a particular set of data.

I do not want to give the impression that the only limitation of FORTRAN-type languages has to do with statistical algorithms. There are a variety of other data processing problems which are commonplace in the social sciences which can be extremely tedious to program in FORTRAN. One example is the problem of missing observations. These occur whenever a variable cannot be measured for a given subject or some other unit of analysis. There are a variety of standardized procedures which a social scientist follows to deal with this problem but no general language that I know of has implemented them. Another problem has to do with a variety of farily common transformations which social scientists apply to their variables but which do not exist in the FORTRAN-type languages. Perhaps the most common of these is what we call "recoding" of an original measurement. This involves transforming the original values of a variable into some arbitrary set of new values, as when we want to collapse a continuous-type variable into one with a small number of values or categories. Although these kinds of transformations can be coded in a FORTRAN-type language, they can frequently lead a fairly lengthy and repetitive program. As we shall illustrate, a recoding operation in Data-Text stands in the same relationship to the necessary FORTRAN steps as an algebraic formula in FORTRAN stands in relation to the necessary steps in a machine language. The same can be said for a number of other transformational capabilities commonly available in the more popular social science languages. Finally, there are a number of other data processing and routine clerical problems which are not easy to handle without special languages. Included are full labeling of variables and their values (or categories), input of multiple-punch data, selecting or sampling sub-sets of the input data, and controls to classify the input data into an arbitrary number of sub-groups.

The earliest attempts to solve some of these problems for the social scientists led to the development of "canned" programs. By "canned" program I mean one written—usually in FORTRAN—to perform a particular statistical analysis, but written so that it can be used over and over again by different researchers with different data. A user prepares "control" or "parameter" cards which specify the necessary information about his data (e.g., number of variables) and any options which are available; the control cards are used by the program to read in the data properly and to produce the desired results. Canned programs are still used widely by social scientists today, and for some applications canned programs continue to be the most efficient computational solution.

As canned programs became a popular method of computer analysis, collections or "packages" were put together at various locations (usually university research centers) and distributed to researchers at other locations. The oldest, most well-known and widely used of these collections is the BMD series, started by Dixon and his associates at UCLA.[3] The BMD collection for the IBM System/360 contains over 40 programs for almost every type of standard statistical analysis found in applied statistics textbooks. Another collection for the IBM System/360 is the OSIRIS package.[4] Developed at the University of Michigan, OSIRIS represents a significant advance in that the canned programs are "integrated" into one large system which recognizes a "standardized" data file. A file of data can be described and then any number of different statistical routines can be requested which use that file.

The canned program approach still leaves many problems unsolved for the social scientist. While canned programs have largely solved the problem of knowing complicated statistical algorithms, little relief is gained for the many other data processing problems which I have mentioned (special transformations, missing observations, labeling, and the like). Moreover, two new problems have surfaced. Both of these problems result from the fact that most canned program developers are not professional programmers, and therefore they are usually more interested in the statistical algorithm than in program elegance.

The first problem is that all kinds of arbitrary restrictions are placed upon various parameters in the problem—the number of variables allowed, the number of subjects, the number of values or categories of variables, the number of analyses, etc. I wonder how many hundreds of researchers have had a 110-variable factor analysis problem with a program which would accept only 100 variables; or a cross-tabulation requiring 60 cells with a program limited to 50? This would be like a FORTRAN programmer being limited to a total of 5 DO-loops or to arbitrary subscript limits of 50. Of course, any language has some limitations, but all good compilers do dynamic core allocation so that limits are encountered only when available memory is exhausted. Most canned program limits are due to programmer short-cuts such as fixed dimension statements.

Second, little effort is given to making the control cards "readable" to the analyst in the sense that FORTRAN is readable to a programmer. Most control cards consist of arbitrary numeric codes in fixed-format positions. It is practically impossible to remember the set-up procedures from one run to another without a program write-up literally in front of you. For the data analysts, having to "read" these control cards is somewhat analogous to a FORTRAN programmer having to read a hexadecimal dump in order to interpret the steps in his program.

THE DATA-TEXT SYSTEM

The Data-Text system was designed to solve some of these problems and limitations of the package approach. Data-Text is designed to be a "natural" language for the social scientist, just as FORTRAN is a natural language for those familiar with mathematical expressions. In this sense, I think Data-Text represents a further stage in the evolution of "problem-oriented" computer languages. With Data-Text, the social scientist can request complex data analyses as easily as the engineer can write complex formulas in FORTRAN. The first version of the Data-Text system was designed by Couch and others for the IBM 7090/94 series, and it became operational in 1963-64.[5] This version became widely used at more than 20 universities and research centers during the middle 1960s. A substantially revised version of Data-Text has been designed and programmed for third generation computers and a version for the IBM System/360 series became operational in the spring of 1971.*

It should be mentioned that in the latter part of the 1960s other computer systems with similar design goals appeared. One example is the Statistical Package for the Social Sciences (SPSS), created by Nie and his associates at Stanford University and the University of Chicago.[6] Other social scientists have begun experimenting with interactive (or "time-sharing") data analysis systems, such as Meyers' IMPRESS system at Dartmouth College,[7] Miller's DATANAL system at Stanford,[8] the ADMINS system at MIT,[9] Shure's TRACE system at SCD and UCLA,[10] the Brooking's Institution BEAST,[11] and Kuh's TROLL system.[12]

The unique aspects of the Data-Text system in relation to other computer systems is its ability to handle very complicated data processing problems and extremely intricate statistical analyses with a minimum of technical knowledge about the computer and a minimum of familiarity with statistical formulas and algorithms. The requirements for a user are: (1) that he have a concrete research problem which demands a standard statistical analysis of data; (2) that the data are recorded in some systematic format on IBM punched cards (or certain other media, such as magnetic tape or disc); (3) that he understands the type of statistical analysis required for his problem; and (4) that he knows the names of the appropriate statistical analysis procedures. Given such a situation, Data-Text can be used for defining or transforming input variables, for giving them appropriate descriptive labels, and for requesting a great variety of different statistical analyses. The variable definitions and statistical analysis requests are made in a flexible, easy-to-learn "natural" language which makes heavy use of terms familiar to most social science researchers. Data processing features include automatic missing observation procedures, input and manipulation of multipunched data, subgrouping controls, and many others.

A sample of the Data-Text language is probably the easiest way to introduce this idea and to show how Data-Text differs from FORTRAN-type languages. Assume that a researcher has collected data from a group of several hundred college students (the actual number is not important). The data might include measurements of age, sex, race, father's education, family income, college class, an ability test score, expected work after college, several yes-no questions regarding current political-social issues, and a series of questions about school activities. We will assume that the non-numeric measures (for example, sex) have been given codes of some kind (for example, 0 for male, 1 for female) and that the data has been punched onto IBM cards with two cards for each person. The Data-Text instructions shown in Figure 1 define the set of original variables, derive some new variables as transformations

* The original Data-Text system was supported by grants from the National Science Foundation (#GS287 & GS1424). The revised Data-Text system was made possible by a grant from the National Institutes for Mental Health (#MH15884) and by donated computer time from the Harvard Computing Center.

```
Col. 1                                                                                    Col. 80
   *DECK COLLEGE STUDENT STUDY
   *READ CARDS
   *CARD(1-2)FORMAT/ UNIT=COL(1-4),CARD=COL(5)
   *AGE=          COL( 5- 6/1)    = AGE OF STUDENT
   *SEX=          COL( 7/1)       = STUDENT'S SEX(0=MALE/1=FEMALE)
   *RACE=        ACOL( 8/1)       = RACIAL BACKGROUND(B=BLACK/W=WHITE/O=OTHER)
   *INCOME=       COL(10-14/1)    = FAMILY INCOME
   *CLASS=        COL(15/1)       = COLLEGE CLASS(1=FRESH/2=SOPH/3=JUNIOR/4=SENIOR)
   *ABILITY=      COL(27-28/1)    =SCHOLASTIC APTITUDE TEST
   *ITEM(1-4)=    COL(31-34(4)/1) = WILLING TO BE DRAFTED(1=YES/2=NO)/                          *
   *                                APPROVE OF SDS POLICIES()/                                  *
   *                                LIKE NIXON()/                                               *
   *                                LIKE AGNEW()
   *ACTIVITY(1-50)= COL(6-55(50)/2)= SCHOOL ACTIVITIES(0=NO/1=YES)
   *VAR (1)=     COL( 9/1)        = FATHER'S EDUCATION(1=SOME HS/2=HS/3=SOME COLL/              *
   *                                                             4=BA/5=BEYOND BA)
   *VAR (2)=     COL(21/1)        = FUTURE PLANS(0=GRAD SCHOOL/1=PROF SCHOOL/                   *
   *                                             2=BUSINESS/3=TEACHING/4=OTHER/                 *
   *                                             5=DON'T KNOW)
   *VAR(3)=      MEAN ITEM(1,3-4)  = PRO-ESTABLISHMENT SCALE
   *INDEX=       SUM ACTIVITY(1-50)=ACTIVITIES INDEX
   *VAR(4)=      1 IF VAR(1)=4,5 AND INCOME GREATER THAN 20000 = SOCIAL CLASS INDEX             *
   *                                 (1=UPPER/2=MIDDLE/3=LOWER)
   *OR=          3 IF VAR(1)=1,2 AND INCOME LESS THAN 10000
   *OR=          2
   *VAR(5)=      RECODE(A) VAR(2)= FUTURE PLANS II(1=SCHOOL/2=WORK)
   *CODE(A)=     (0,1=1/2-4=2/5=BLANK)
   *VAR(6)=      (ABILITY/AGE)*100 = AGE-GRADED ABILITY
   *CHANGE(1-10)= POSTEST(1-10)-PRETEST(1-10)
   *WRITE DATATEXT DISC 4
   *COMPUTE STATISTICS(AGE, INCOME, ITEM, VAR(6)), SKEW
   *COMPUTE T-TESTS(ABILITY, VAR(3)), GROUP BY SEX
   *COMPUTE FREQUENCIES
   *COMPUTE CORRELATIONS(ACTIVITY),TEST
   *COMPUTE CROSSTABS(VAR(2), ITEM BY SEX, RACE, CLASS), CHI SQUARE
   *COMPUTE REGRESSION(ABILITY ON AGE, SEX, VAR(1), INCOME), GROUP BY CLASS
   *COMPUTE ANOVA(SEX BY RACE BY VAR(4)), ABILITY
   *START
```

Figure 1—A sample data-text program

of the original set, and request several types of statistical analyses. (In many cases these instructions are punched onto IBM cards, one instruction per card. In other instances, they might be lines typed at a remote typewriter console.)

The instructions fall into several types. The instruction

```
*DECK COLLEGE STUDENT STUDY
```

is a header instruction which signifies the start of a set of Data-Text instructions and which provides a title that will appear on every page of computer print-out.

The next two instructions relate to I/O operations:

```
*READ CARDS
*CARD(1-2)FORMAT/UNIT=COL(1-4),
                  CARD=COL(5)
```

The READ identifies the mode of data input (punched cards), while the FORMAT provides information about the number of cards per student and some special identification fields. The term CARD(1-2) indicates 2 cards per subject; the term UNIT refers to a field in each card for identifying the unit of analysis (such as a subject number); the term CARD refers to a field for identifying the different data cards for each UNIT. Thus, this example indicates two cards per UNIT, a subject number in columns 1 to 4 of each card, and a card number in column 5. Although there is some syntactic similarity to FORTRAN in these two instructions, their effect is quite different. First, no looping controls are required around the READ instruction; all data cards will be processed automatically according to the other operations specified in the program.* Second, the FORMAT instruction in this ex-

* A special *SELECT . . . IF instruction is available for selecting sub-sets of the data file; see Figure 2.

ample only provides information about the number of cards per subject and the identification fields; variable fields are defined in separate instructions. Moreover, if the UNIT and CARD fields are specified as in the example, the cards for each subject are automatically checked for consistent subject numbers and card sequence (taking the first subject as the prototype). These checking operations are extremely important for some of the large-scale survey research studies.

The next set of instructions define the variables to be used in the statistical analysis requests. For example, the instruction

```
*SEX=COL(7/1)
     =STUDENT'S SEX(0=MALE/1=FEMALE)
```

defines a variable named SEX which appears in column 7 of card 1 for each subject. It also gives a user-supplied descriptive label and, within the parentheses, it describes the values actually punched in column 7 and relates them to user-supplied category labels (e.g., 0= MALE means that 0's were punched to represent males). The ability to specify labels and values enables an analyst to construct a true "machine readable" codebook for his data set. Moreover, the use of the variable and category labels is not confined to the instruction listing; they are also used in all computer print-out of statistical results which use the variables and/or their categories.

Alphabetic-coded variables can be indicated by preceding the COL-specification with the letter "A":

```
*RACE=ACOL(8/1)
      =RACIAL BACKGROUND
    (B=BLACK/W=WHITE/O=OTHER)
```

This definition indicates a variable which was coded in column 8 with the letter values B, W, and O.

Entire arrays of variables can also be defined without the use of explicit looping controls, as in the example

```
*ACTIVITY(1-50)=COL(6-55(50)/2)
               =SCHOOL ACTIVITIES
               (0=NO/1=YES)
```

In this case the variable and value descriptions will apply to all 50 variables in the array. The example *ITEM(1-4) shows how several variables can be defined with the same value descriptions but different variable labels.

The example in Figure 1 illustrates several of the special transformation capabilities of Data-Text. While the logical and arithmetic transformations used to create VAR(4) and VAR(6) resemble those available in most languages, the special functions MEAN, SUM and RECODE used in defining VAR(3), INDEX, and VAR(5) are unique. The MEAN and SUM are "summary" functions which operate *across* the variables for a given subject, so that an average score (for the MEAN) and a count (for the SUM) are derived for each subject in the sample. Again, no looping controls are necessary for these operations.

The RECODE function embodies a great deal of power. It works in conjunction with a code statement (*CODE(A) in the example) which specifies the recoding equations. Each equation consists of a series of original values together with the new value they are to be assigned in the transformed variable. Thus, only two instructions are required to transform one set of values into any combination of new values. Although this kind of operation is programmable in FORTRAN, it requires a great many assignment, IF and GO TO statements.

One of the equations in the *CODE statement is 5=BLANK. The term BLANK is a special constant in Data-Text which denotes a missing observation. This means that a subject with a score of 5 in VAR(2) will be treated as missing in VAR(5). BLANK values are automatically assigned to variables for subjects with blank columns in the input data fields. That is, if columns 5 and 6 in card 1 are blank for a given subject, then a BLANK value is automatically assigned to the AGE variable for that case. Regardless of where BLANK values come from, they are handled automatically in all transfomation and statistical routines. A series of default rules exist which are appropriate to the type of transformation or the type of statistical procedure. For example, BLANK's are ignored by the MEAN function in the VAR(3) instruction, so that the average is taken only of non-BLANK values. In the VAR(6) instruction, however, which uses a regular arithmetic expression, the opposite is true: if either of the arguments ABILITY or AGE is BLANK for a subject, then the result is BLANK. In most of the statistical routines indicated by the COMPUTE instructions, the default is to ignore BLANK values; but options exist which will do other things (like excluding a subject from the analysis if any variable has a BLANK value). This automatic processing of missing observations has proved to be an extremely valuable feature in social science applications.

Before turning to the statistical controls, another instruction should be noted. The instruction

```
*CHANGE(1-10)=POSTEST(1-10)
             -PRETEST(1-10)
```

illustrates what we call a "list" expression. The subtraction operator is applied element-wise to the two argument lists, so that each variable in the CHANGE array is the difference between the POSTEST and PRETEST variable in the corresponding list position.

```
*DECK COLLEGE STUDENT STUDY—SECOND RUN
*READ DATATEXT DISC 4
*LEVEL=ORDER ABILITY=ABILITY LEVEL(0-9=LOW/10-24=MIDDLE/OVER 24=HIGH)
*SELECT UNIT IF SEX= 0
*COMPUTE CROSSTABS(LEVEL BY RACE, CLASS), CHI SQUARE
*START
```

Figure 2—Using a data-text file

Lists of variables can be combined into any arbitrary arithmetic expression, and the expression is evaluated for each element in the list. This provides a very powerful transformational ability without the necessity of looping controls.

The COMPUTE instructions are commands for statistical analyses on the variables defined in the preceding instructions. The statistical procedure is requested by name, such as T-TEST or CROSSTAB or CORRELATION. Generally speaking, the only parameters required are the lists of variables to be included in the analysis and any special options which are desired. The instruction

```
*COMPUTE STATISTICS(AGE, INCOME,
    ITEM, VAR(6)), SKEW
```

will compute means, standard deviations, and the number of nonmissing observations for each variable in the list across the whole sample of students; the option SKEW will also cause a measure of skewness to be computed. The instruction

```
*COMPUTE CORRELATIONS(ACTIVITY),
    TEST
```

will cause a 50 by 50 matrix of product-moment correlations to be computed for the array ACTIVITY(1-50), and the TEST option will result in a statistical test of significance for each correlation.

While there is some resemblance here between the COMPUTE instruction and a subroutine call in many languages, the analogy cannot be pushed too far. First, the "arguments" include reference only to the variable names and not to the many other quantities and arrays which are used by the routine, such as variable labels, value description, number of categories, and so forth. Second, and more important, some of the statistical requests have special key words embedded in the variable list which must be encoded by a special compiler tailored to each statistical procedure.

For example, in the instruction

```
*COMPUTE REGRESSION(ABILITY ON AGE,
    SEX, VAR(1), INCOME),
    GROUP BY CLASS
```

the key word ON indicates the independent variables (AGE, SEX, VAR(1), INCOME) on which the dependent variable (ABILITY) is to be regressed. The GROUP option indicates the analysis is to be carried out on the four college class subgroups defined by the CLASS variable. In the COMPUTE ANOVA instruction,*

```
*COMPUTE ANOVA(SEX BY RACE BY
    VAR(4)), ABILITY
```

the key word BY indicates a 3-way factorial analysis of variance design with ABILITY as the dependent or criterion measure. In other words, the COMPUTE instructions are also expressed in a natural language which contain operators and a syntax meaningful to each particular analysis.

All of the statistical routines are almost literally without limitations. Correlation matrices are not limited to available core; a cross-tabulation can be *n*-ways, with no limit to *n*; a variable used for classification purposes in a cross-tabulation, an anova, or in the GROUP BY option can have any number of categories; and if all the analyses requested will not fit in available core, the data is automatically saved in internal form so that repeated passes can be made over the data to satisfy all the requests. These features reduce the frustrations often encountered with other statistical packages which have established arbitrary parameter limitations.

It is rarely the case that an analyst will obtain all of his statistical results in a single run. Sometimes the same data set will be analyzed several dozen times. Data-Text offers a method for saving all of the defined variables, labels, and data in a standardized binary file. Since no compilation, transformation, and data conversion needs to be done, the time required to process this file is substantially less than the time it takes to process the original raw data.* In our sample program, the instruction

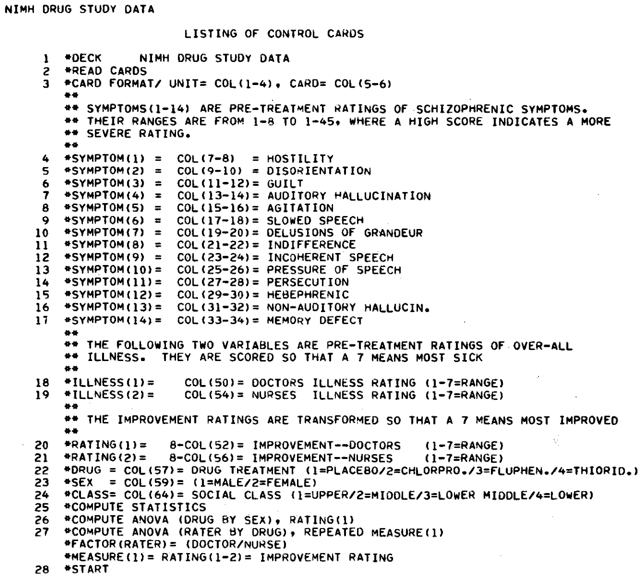
```
*WRITE DATATEXT DISC 4
```

creates a standardized file for this set of data.

Input of a DATATEXT file does not preclude the use of the full transformational language. New vari-

* The term ANOVA is a fairly standard abbreviation for "analysis of variance."

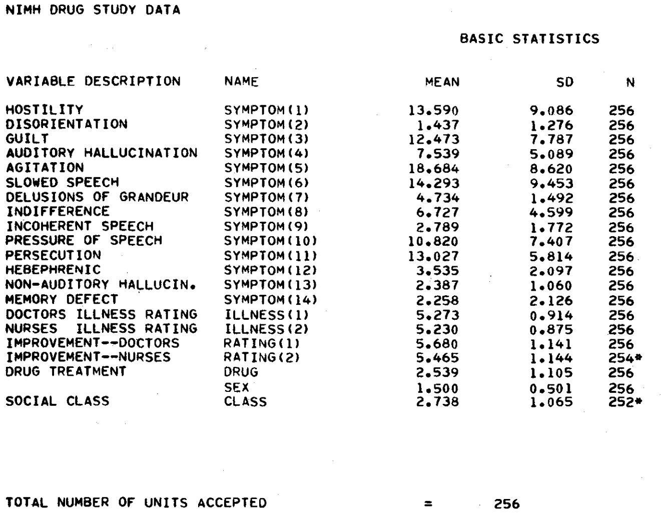
* The savings range from 50 percent to 90 percent of CPU time, depending upon the size of the data file (the larger the file, the greater the savings).



```
NIMH DRUG STUDY DATA

                         LISTING OF CONTROL CARDS

  1  *DECK     NIMH DRUG STUDY DATA
  2  *READ CARDS
  3  *CARD FORMAT/ UNIT= COL(1-4), CARD= COL(5-6)
     **
     ** SYMPTOMS(1-14) ARE PRE-TREATMENT RATINGS OF SCHIZOPHRENIC SYMPTOMS.
     ** THEIR RANGES ARE FROM 1-8 TO 1-45, WHERE A HIGH SCORE INDICATES A MORE
     ** SEVERE RATING.
     **
  4  *SYMPTOM(1) =  COL(7-8)   = HOSTILITY
  5  *SYMPTOM(2) =  COL(9-10)  = DISORIENTATION
  6  *SYMPTOM(3) =  COL(11-12)= GUILT
  7  *SYMPTOM(4) =  COL(13-14)= AUDITORY HALLUCINATION
  8  *SYMPTOM(5) =  COL(15-16)= AGITATION
  9  *SYMPTOM(6) =  COL(17-18)= SLOWED SPEECH
 10  *SYMPTOM(7) =  COL(19-20)= DELUSIONS OF GRANDEUR
 11  *SYMPTOM(8) =  COL(21-22)= INDIFFERENCE
 12  *SYMPTOM(9) =  COL(23-24)= INCOHERENT SPEECH
 13  *SYMPTOM(10)=  COL(25-26)= PRESSURE OF SPEECH
 14  *SYMPTOM(11)=  COL(27-28)= PERSECUTION
 15  *SYMPTOM(12)=  COL(29-30)= HEBEPHRENIC
 16  *SYMPTOM(13)=  COL(31-32)= NON-AUDITORY HALLUCIN.
 17  *SYMPTOM(14)=  COL(33-34)= MEMORY DEFECT
     **
     ** THE FOLLOWING TWO VARIABLES ARE PRE-TREATMENT RATINGS OF OVER-ALL
     ** ILLNESS.  THEY ARE SCORED SO THAT A 7 MEANS MOST SICK
     **
 18  *ILLNESS(1)=    COL(50)= DOCTORS ILLNESS RATING (1-7=RANGE)
 19  *ILLNESS(2)=    COL(54)= NURSES  ILLNESS RATING (1-7=RANGE)
     **
     ** THE IMPROVEMENT RATINGS ARE TRANSFORMED SO THAT A 7 MEANS MOST IMPROVED
     **
 20  *RATING(1)=   8-COL(52)= IMPROVEMENT--DOCTORS   (1-7=RANGE)
 21  *RATING(2)=   8-COL(56)= IMPROVEMENT--NURSES    (1-7=RANGE)
 22  *DRUG = COL(57)= DRUG TREATMENT (1=PLACEBO/2=CHLORPRO./3=FLUPHEN./4=THIORID.)
 23  *SEX  = COL(59)= (1=MALE/2=FEMALE)
 24  *CLASS= COL(64)= SOCIAL CLASS (1=UPPER/2=MIDDLE/3=LOWER MIDDLE/4=LOWER)
 25  *COMPUTE STATISTICS
 26  *COMPUTE ANOVA (DRUG BY SEX), RATING(1)
 27  *COMPUTE ANOVA (RATER BY DRUG), REPEATED MEASURE(1)
     *FACTOR(RATER)= (DOCTOR/NURSE)
     *MEASURE(1)= RATING(1-2)= IMPROVEMENT RATING
 28  *START
```

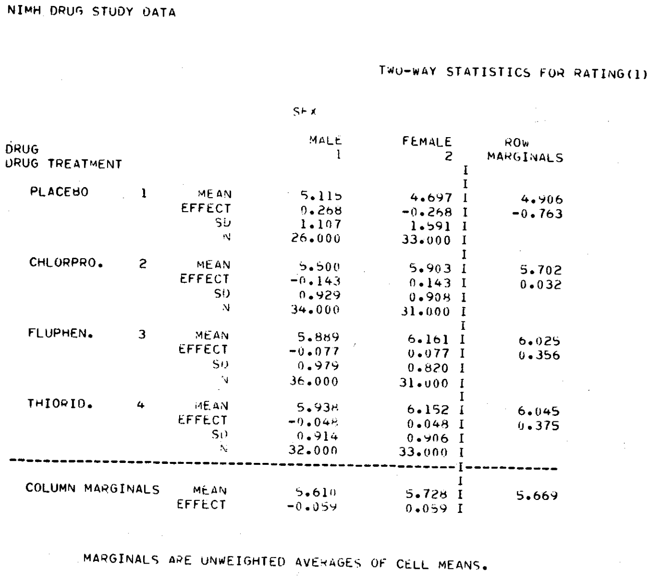
Figure 3—Sample output from a data-text run

ables can be derived from existing variables just as in a regular run. Figure 2 shows a run using the DATATEXT file created in the first example. A new variable, LEVEL, is derived by collapsing the existing variable ABILITY using the special collapsing function ORDER. Some new cross-tabulations are requested, and the SELECT instruction will restrict processing to the sub-group of males. It is clear from this example that using a DATATEXT file results in a much shorter and simpler program.



```
NIMH DRUG STUDY DATA

                                                BASIC STATISTICS

VARIABLE DESCRIPTION       NAME                  MEAN            SD       N

HOSTILITY                  SYMPTOM(1)          13.590         9.086     256
DISORIENTATION             SYMPTOM(2)           1.437         1.276     256
GUILT                      SYMPTOM(3)          12.473         7.787     256
AUDITORY HALLUCINATION     SYMPTOM(4)           7.539         5.089     256
AGITATION                  SYMPTOM(5)          18.684         8.620     256
SLOWED SPEECH              SYMPTOM(6)          14.293         9.453     256
DELUSIONS OF GRANDEUR      SYMPTOM(7)           4.734         1.492     256
INDIFFERENCE               SYMPTOM(8)           6.727         4.599     256
INCOHERENT SPEECH          SYMPTOM(9)           2.789         1.772     256
PRESSURE OF SPEECH         SYMPTOM(10)         10.820         7.407     256
PERSECUTION                SYMPTOM(11)         13.027         5.814     256
HEBEPHRENIC                SYMPTOM(12)          3.535         2.097     256
NON-AUDITORY HALLUCIN.     SYMPTOM(13)          2.387         1.060     256
MEMORY DEFECT              SYMPTOM(14)          2.258         2.126     256
DOCTORS ILLNESS RATING     ILLNESS(1)           5.273         0.914     256
NURSES  ILLNESS RATING     ILLNESS(2)           5.230         0.875     256
IMPROVEMENT--DOCTORS       RATING(1)            5.680         1.141     256
IMPROVEMENT--NURSES        RATING(2)            5.465         1.144     254*
DRUG TREATMENT             DRUG                 2.539         1.105     256
                           SEX                  1.500         0.501     256
SOCIAL CLASS               CLASS                2.738         1.065     252*



TOTAL NUMBER OF UNITS ACCEPTED                  =    256
```

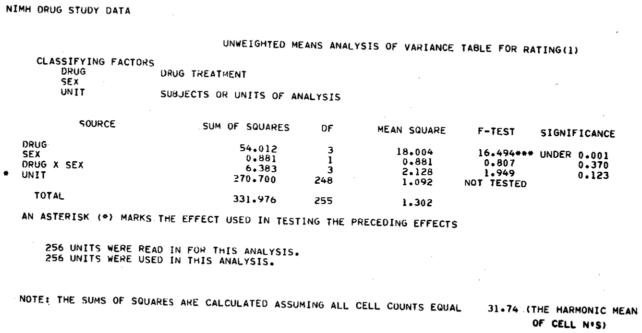
Figure 3b



```
NIMH DRUG STUDY DATA


                                         TWO-WAY STATISTICS FOR RATING(1)

                              SEX

                                  MALE        FEMALE       ROW
DRUG                                 1             2     MARGINALS
DRUG TREATMENT                                         I
                                                       I
  PLACEBO      1        MEAN     5.115         4.697 I     4.906
                      EFFECT     0.268        -0.268 I    -0.763
                          SD     1.107         1.591 I
                           N    26.000        33.000 I
                                                       I
  CHLORPRO.    2        MEAN     5.500         5.903 I     5.702
                      EFFECT    -0.143         0.143 I     0.032
                          SD     0.929         0.908 I
                           N    34.000        31.000 I
                                                       I
  FLUPHEN.     3        MEAN     5.889         6.161 I     6.025
                      EFFECT    -0.077         0.077 I     0.356
                          SD     0.979         0.820 I
                           N    36.000        31.000 I
                                                       I
  THIORID.     4        MEAN     5.938         6.152 I     6.045
                      EFFECT    -0.048         0.048 I     0.375
                          SD     0.914         0.906 I
                           N    32.000        33.000 I
-----------------------------------------------------I-----------
                                                       I
 COLUMN MARGINALS       MEAN     5.610         5.728 I     5.669
                      EFFECT    -0.059         0.059 I


        MARGINALS ARE UNWEIGHTED AVERAGES OF CELL MEANS.
```

Figure 3c

As I have noted, the use of labeling specifications illustrated in Figures 1 and 2 is not confined to a program listing. All of the Data-Text statistical routines use the labels to produce readable output. Since the sample programs were hypothetical, we cannot show output for them. However, Figure 3 illustrates partial output from an actual run using real data. The Data-Text program which defines the variables and requests the analyses is shown on the first page of output.* The remaining pages show the output resulting from the STATISTICS and ANOVA instructions.



```
NIMH DRUG STUDY DATA

                      UNWEIGHTED MEANS ANALYSIS OF VARIANCE TABLE FOR RATING(1)
    CLASSIFYING FACTORS
        DRUG          DRUG TREATMENT
        SEX
        UNIT          SUBJECTS OR UNITS OF ANALYSIS

        SOURCE                SUM OF SQUARES    DF     MEAN SQUARE    F-TEST    SIGNIFICANCE

   DRUG                           54.012         3        18.004     16.494*** UNDER 0.001
   SEX                             0.881         1         0.881      0.807          0.370
   DRUG X SEX                      6.383         3         2.128      1.949          0.123
*  UNIT                          270.700       248         1.092    NOT TESTED

     TOTAL                       331.976       255         1.302

   AN ASTERISK (*) MARKS THE EFFECT USED IN TESTING THE PRECEDING EFFECTS

      256 UNITS WERE READ IN FOR THIS ANALYSIS.
      256 UNITS WERE USED IN THIS ANALYSIS.


   NOTE: THE SUMS OF SQUARES ARE CALCULATED ASSUMING ALL CELL COUNTS EQUAL   31.74 (THE HARMONIC MEAN
                                                                                  OF CELL N'S)
```

Figure 3d

* The instructions which begin with double asterisks (**) are comment cards.

LIMITATIONS OF THE DATA-TEXT SYSTEM

While a computer system may look good on paper, any real implementation always involves sacrifices and compromises. Some of the more obvious limitations and deficiencies involve the actual number of features which are available. Our investment of time and energy in making certain features as flexible as possible meant that other features could not be developed. In the transformational language, for example, there are no DO-loop and variable subscript capabilities. While the list expressions and summary functions available in Data-Text make these features less crucial than in other languages, on occasion a very complex transformation is desired for which these controls would be useful. Perhaps the best way to solve this would be to permit a subroutine call to a regular FORTRAN routime. The addition of such a feature would substantially enhance the transformational power of Data-Text (or any other social science language).

On the statistical side, we have confined the system to what might be called "standard" statistical routines that enjoy a wide usage in the social sciences. Some of the packaged program collections mentioned earlier (BMD, OSIRIS) offer many more statistical routines (discriminant function analysis, canonical correlation, Guttman scaling, etc.). This limitation is not a fatal one, however, since in Data-Text it is possible to write out a new transformed data set which can be fed into these other routines. But it would be more convenient to have all statistical analysis within the same integrated system.*

One of the major goals of the revised Data-Text system is machine independence. Therefore, almost all coding was done in a standard subset of FORTRAN IV. This fact coupled with the heavy overhead common to all large systems means that Data-Text is fairly slow and bulky. The load module requires about 200 tracks of an IBM 2314 disc, and although it has been extensively overlaid, the minimum region size is about 200K (250K is sufficient to run most jobs). Regarding speed, Data-Text is far more expensive to run than a FORTRAN program written to perform the same specific task. For example, the run illustrated in Figure 3 required 637 accesses and 20 seconds of CPU time on an IBM 360/65. In determining cost effectiveness, of course, one must take into account the time to do the programming. An argument about the cost of Data-Text versus FORTRAN is not dissimilar to the same one about FORTRAN versus assembler language.

Another implication of machine independence is that we were confined to a basic subset of special symbols. This is one of the reasons (although not the only one) why our syntax may look strange to some language designers. The symbols =) / (are used heavily as separators and sometimes with different meaning. Slashes mean division in arithmetic statements, but they are also used to separate different category labels and recoding transformations.* Another reason for "strange" syntax has to do with maintaining upward-compatibility with the original Data-Text system. In the original system there was no *START instruction to separate instruction cards from data cards; instead, instructions were recognized by an * in column 1. Since we tried to keep the revised version compatible, we inherited this feature.

COMPARISON WITH OTHER SYSTEMS

I have noted that there are other systems developed for social science data analysis which are similar to Data-Text. Accordingly, there is a great deal of interest in the potential user community in the differences among the systems. While there is not sufficient room to present a detailed comparison here, some of the major differences can be summarized. I must state at the outset that I am not an unbiased observer in this regard, and knowledgeable users may well have different opinions.

Since Data-Text was developed primarily as a batch system, the biggest differences are with interactive systems. We can take IMPRESS[7] as a popular representative. The main advantage of IMPRESS is that its command structure assumes that a student or researcher is in front of a console requesting one kind of analysis at a time. The result is that a great deal of communication can occur to help guide the analysis. This is extremely valuable for teaching purposes. For the researcher doing large-scale runs on his own data, however, most time-sharing systems (including IMPRESS) lack a variable definition and labeling capability which works on raw data. The raw data is put into a standardized, labeled form as a *separate* step (and without a language); in Data-Text this step is integrated with the transformational and statistical language. Also, most time-sharing systems are understandably less concerned about fixed parameter limits.

* The Cambridge Project under way at MIT and Harvard represents one very comprehensive attempt to do this on a time-sharing system.

* Commas could not have been used without making certain expressions look confusing; for example, consider (1 = 2/2, 3, 5 = 1) on a CODE statement.

Space is generally at a premium, and it is not uncommon to find fairly low limits on the number of cells in a table, the number of variables in a regression, and so forth.

The package collections such as BMD[3] and OSIRIS[4] are also quite different from Data-Text. For one thing, they offer many more statistical or analytic routines than Data-Text. Also, since they are really sets of individual programs, there is no system overhead and jobs can generally be executed in a smaller region and with much less CPU time than the same jobs in Data-Text. On the other hand, the lack of integration means that a lot of information has to be repeated for each separate analysis; that the control cards for different routines often have different syntax; and that, like time-sharing systems, variable definitions and transformations using raw data comprise a step *separate* from the analysis.

The system most similar to Data-Text is SPSS. Both are batch-oriented; both are integrated so that variable definition, transformation, and statistical analysis instructions are given in the same program; and both overlap considerably in their analytic routines.

Aside from syntax, the major differences in the transformational language have to do with speed versus power. Unlike Data-Text, SPSS does not have list expression or summary function capability, and it does not handle missing observations automatically in all variable transformations. This means that a SPSS program will generally require many more statements than Data-Text if there are conditions which require these features. On the other hand, for simple problems which do not need these features SPSS will take much less CPU time.

While analytic routines such as cross-tabulation, regression, and factor analysis are fairly similar in both systems, SPSS alone has Guttman scaling, partial correlations, and histograms. Data-Text alone has scatterplots, t-tests, and a generalized analysis of variance routine; the latter makes it possible to request just about any kind of analysis of variance design with a simple language tailored to this statistical procedure. Also, the Data-Text cross-tabulation routine will handle multivalued variables which can arise from multiple-choice response categories in a survey (it can also read in and transform multiple-punched cards). A more important difference for some researchers may be the parameter-free character of the Data-Text routines. The size of the matrix for the Data-Text correlation and factor analysis routines is not limited by available memory. This means that even in relatively small work space (e.g., 70K) a 500-by-500 (or one of any size, for that matter) correlation matrix or factor analysis can be computed.

THE IBM SYSTEM/360 IMPLEMENTATION

A brief amount of information about the implementation of Data-Text will supply perspective. Data-Text is a "compiler/interpreter" system programmed largely in ANSI FORTRAN IV on the IBM System/360 and 370 (a version is also underway for the CDC 6000 series).* During the compile phase the variable definitions and transformations are compiled into an intermediate language, and the statistical COMPUTE requests are encoded into sets of internal signals. During a second phase, the intermediate language is operated on by an interpreter for each unit of analysis to produce the final variables for the statistical routines. In addition, this set of variables is passed to each of the requested statistical routines for accumulation of the necessary quantities according to the encoded signals (e.g., sums, sums of squares, etc.). In the third and final phase each statistical routine computes and prints out their respective results using the accumulations from the second phase and the encoded signals from the compile phase.

The avoidance of fixed parameter limits in statistical routines is achieved in several ways. First, if there are more COMPUTE requests than there is room in available memory to accumulate the necessary quantities, then the data for the variable in question is automatically saved in a special file and subsequent passes are made over the file until all statistical requests are satisfied. Second, the programs for each statistical procedure are written using "dynamic" core allocation. That is, all accumulation arrays are linear, and subscripting is accomplished by use of linear subscript functions. Therefore, each statistical routine will use only as much space as is absolutely necessary for the analysis requested. Finally, certain statistical routines are programmed to use scratch data set space if necessary to process a request that will not fit in available memory. For example, the factor analysis routine will compute and save a correlation matrix in "pieces" if the whole matrix does not fit in available memory; the factoring is likewise computed in a iterative procedure which processes successive "pieces" of correlation matrix.

SUMMARY AND CONCLUSIONS

I began by stating that while there has been a lot of duplication of effort in the field of social science software, much of it was necessary for the normal develop-

* Major parts of the non-statistical systems design and programming has been carried out by Cambridge Computer Associates, Cambridge, Massachusetts.

ment of the field. Perhaps that claim can be better understood now. The standard languages distributed routinely by the hardware manufacturers did not meet many of the special statistical needs of social and behavioral scientists. The result was that almost every university computing center developed a package of canned programs to meet these needs. As the solutions became somewhat standardized, it became possible to develop more comprehensive languages like Data-Text and SPSS. By combining procedures for many different data processing tasks into one integrated language, these systems offer users considerably more power and convenience.

REFERENCES

1 H F CLINE
Preliminary summary of a study of computer uses in social science research
Unpublished manuscript Russell Sage Foundation New York 1970

2 D J ARMOR A S COUCH
The data-text primer
Free Press New York 1972

3 W J DIXON Ed
BMD—Biomedical computer programs
University Press Berkeley California 1967

4 *Inter-university consortium for political research and institute for social research OSIRIS II*
University of Michigan 1968

5 A S COUCH
The data-text system
Unpublished users manual Harvard University 1965

6 N NIE et al
SPSS: Statistical package for the social sciences
McGraw-Hill Book Co New York 1970

7 J A DAVID J H STERNICK
The IMPRESS primer
Dartmouth College 1971

8 J R MILLER III
DATANAL: An interpretative language for on-line analysis of empirical data
Mitre Technical Report MTR-48 Bedford Massachusetts 1967

9 S D McINTOSH D M GRIFFEL
The ADMINS primer
Center for International Studies Massachusetts Institute of Technology Cambridge Massachusetts 1968

10 R P ESADA
TRACE—Model II user's guide
System Development Corporation Santa Monica California 1967

11 J W BEAN et al
The BEAST
The Brookings Institution Washington DC 1968

12 For information write Walt Maling at the NBER Computer Research Center for Economics and Management Science 545 Technology Square Cambridge Massachusetts 02139

LSI—Implications for future design and architecture

by S. F. DENNIS and M. G. SMITH

IBM Thomas J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

We attempt to justify a number of ways in which we believe LSI is most likely to affect systems design and architecture. It is not only the physical componentry of systems that will undergo change, but also our thinking, as an industry, about the goals, methods, uses and people associated with data processing. LSI will enlarge the scope of computer usage, while simultaneously reducing and altering the human effort required to install and use systems.

More specifically, we will review the major factors that motivate and limit the use of LSI. We will examine how LSI will change the distribution of costs in systems, and thereby change our present priorities in design and architecture. We will mention some general hardware functions, and as an illustration of some of these, we will explore how hardware and software might be structured in a large-scale teleprocessing system in the future.

WHY IS LSI EXCITING?

LSI takes many forms, directed to both logic and memory. For our purposes, it is hundreds and thousands of circuits or memory cells per chip. LSI is usually implemented via semiconductor technology, e.g., bipolar or field-effect technology (FET), but conceptually could include the magnetic "bubble" technology.

Probably the most interesting property of LSI is its potential cost. The trade and engineering literature reflects a pattern[1,2,3] which can be reduced to a crude rule-of-thumb cost model. It assumes that the level of integration chosen at any given time is such as to produce a modest chip yield at a cost comparable to the cost to package that level of integration.* We will calibrate our rule-of-thumb model at the \$0.0003/memory cell (bit) recently projected for the field-effect transistor random access memory (FET RAM) by Bob Noyce of Intel.[3] (The authors also believe this is possible.) A plausible "good" wafer cost is projected to be somewhere around \$25. (A "good" wafer is one that passes quality control tests.) Ten good chips might be an acceptable yield per wafer.

We conjecture that LSI costs will break down as follows:

Chip Costs:

$$\frac{\$25/\text{wafer}}{10\text{ good chips/wafer}} \equiv \$2.50/\text{chip}.$$

The part number cost, i.e., the cost to generate and handle a different part type,** must also be included. Thus, assuming comparable a packaging cost, and a part number generation cost term for "N" parts, the overall chip part cost is:

$$\$2.50/\text{chip} + \$2.50/\text{chip package} + \frac{\text{part number cost}}{\text{N}}$$

Therefore, the cost of a circuit, assuming n circuits per chip, is:

$$\frac{1}{\text{n}}\left(\$5.00 + \frac{\text{part number cost}}{\text{N}}\right).$$

By selecting a part number cost, introducing relative densities for the various technologies and the type of part (i.e., logic, Read-Only Memory [ROM], Random Access Memory [RAM], etc.), and estimating production quantities, we can project the costs which form the basis for our considerations.

First, we examine part number cost. If this cost is high, then LSI designs for all but high production parts

* Slices of silicon, called wafers, have patterns of hundreds or thousands of circuits or memory cells which, when the wafer is diced, become chips. These chips are first packaged, typically one to a package, and subsequently interconnected by higher-level packaging into logic or memory subsystems.

** The part number costs are those costs attributed to the uniqueness of a part, starting after the logic design and continuing through the fabrication, testing and handling of the part.

TABLE I—Postulated Device Costs

Semiconductor Hardware	*Circuits or Cells Per Chip*	*Mfg. Cost Per Circuit or Cell in Cents Number of Parts per Part Number*		
		10	*100*	*10,000*
ROS	64,000	0.16	0.023	0.008
RAM (FET)	16,000	0.66	0.09	0.032
RAM (Bipolar)	4,000	2.6	0.38	0.13
Logic (FET)	2,000	5.3	0.75	0.25
Logic (Bipolar)	1,000	10.0	1.5	0.5

will be questionable or prohibitive.[4,5] This might inspire novel design approaches in which like parts could be used. On the other hand, we believe that design automation techniques will bring part number costs eventually to something like $1000, as opposed to tens of thousands of dollars in the past. The pressures to make LSI a tool of the designer rather than his master will certainly bring this about.

The next step is to estimate densities. For example, in random access memory, FET designs have been reported with 4000 cells per chip. In recent years, integration levels have more than doubled every year, and while obstacles in photolithography and packaging certainly exist, there are at least several more doublings likely. (Further, the densities used below are not the most optimistic estimates in the industry.)[3] Based upon a $1000 part number cost and the rule-of-thumb formula given above, we might expect to see the following costs of LSI manufacture (Table I):
These costs could be thought of as representative of maximum yield-density configurations in that pushing performance in any of the five hardware categories would result in higher costs: higher performance logic or memory variations might cost considerably more. In addition, not all components could be expected to be produced in the same quantities; we would expect to see RAM components produced in very high quantities, whereas logic components would usually be produced in small quantities—a notable exception being logic for use in terminals.

IMPLICATIONS FOR SYSTEM COMPONENTS

What do these costs mean to CPU implementation? It has been estimated that there are around 100,000 digital processors in use today. Since most processors are small, let us suppose that there are about 5000 circuits and 20,000 bytes of main memory per machine. For today's processors with tomorrow's manufacturing costs of 0.5¢ a circuit and 0.03¢ per bit, the total logic and memory requirements for 100,000 processors would cost only $2.5 million for logic and only about $5 million for memory. Additional perspective is gained by examining what these costs would do to a "typical" large system. In Table II, we define a hypothetical contemporary system and apply costs to the various components typical of those in industry today. Then we reprice these components in terms of the conjectured LSI prices and our best guesses about the other components.[5] Smaller reductions are assumed in other than memory and logic costs, since we do not foresee technological breakthroughs in those costs.

Although the late 70's cost of some of the key componentry could be one-fiftieth that of today's cost, the overall cost ratio (today:late 70's) is about 6 to 1. With peripheral equipment added in, the ratio might be more nearly 3:1.

Cost of ownership

But this view ignores non-hardware expenses. If we look at system cost as classic "cost of ownership" (Figure 1), we see that the static costs of planning, programming and operations bring the cost reduction to only 28 percent. Of course, 28 percent is not to be callously ignored.

However, even "cost of ownership" is not the total cost. For example, data acquisition, its conversion to machine-readable form, and preparation of machine output for an ultimate consumer's use may cost a substantial amount in comparison with "cost of ownership." Further, the increasing complexity of our present

TABLE II—A Hypothetical Large Processor

	Today		Late 70's	
	Cost	*Percent of System Cost*	*Cost*	*Percent of System Cost*
Main Memory 1M byte	$120K	32	$ 2.7K	4
Special Memories	40K	11	2.0K	3
Logic	70K	19	1.2K	2
Special Circuits	25K	7	10.0K	15
Higher Level Packaging	35K	9	10.0K	15
Power Supplies & Cooling	40K	11	20.0K	30
Other	40K	11	20.0K	30
	$370K	~100	$65.9K	~100

systems design is adding to the costs of installing, extending and maintaining systems and of adding new applications. So, even the 28 percent is an overly optimistic estimate.

Cost of terminals

Since terminal costs will become steadily more important, we need to try to forecast them.[6] We define a hypothetical terminal, in this case a display terminal,[7] and assume the hardware required, along with prices retrieved from trade journals, circa 1970. Again, by applying our rule-of-thumb LSI projections to LSI parts, and lesser reductions to other components, we have approximately a 3-to-1 reduction, as shown in Table III.

Clearly, significantly more logic and memory function could be supplied at a modest additional cost, providing the user with more terminal function per dollar, rather than merely with lower cost components *per se.*

TABLE III—Display Terminal Cost Conjecture

CRT Head	$ 75	$ 60
Character Generator	50	2
Logic (1200 Circuits)	240	6 (.5¢/ckt)
Storage 10K	90	5 (.05¢/cell)
Special Circuits	100	50
Higher Level Packaging	50	25
Power	200	80
Keyboard	150	80
Other	145	86
TOTAL	$1100	~$400

GENERAL SYSTEMS IMPLICATIONS OF LSI COSTS

It is evident that LSI moves the major cost component of computer systems away from CPU-Memory and toward other system components. What effect should this have on our techniques? Novel systems or architectural approaches probably are now more feasible, but conventional approaches are also greatly enhanced. Evolutionary changes are certainly easiest to project—but also should be expected. As we see it, the general implications are these:

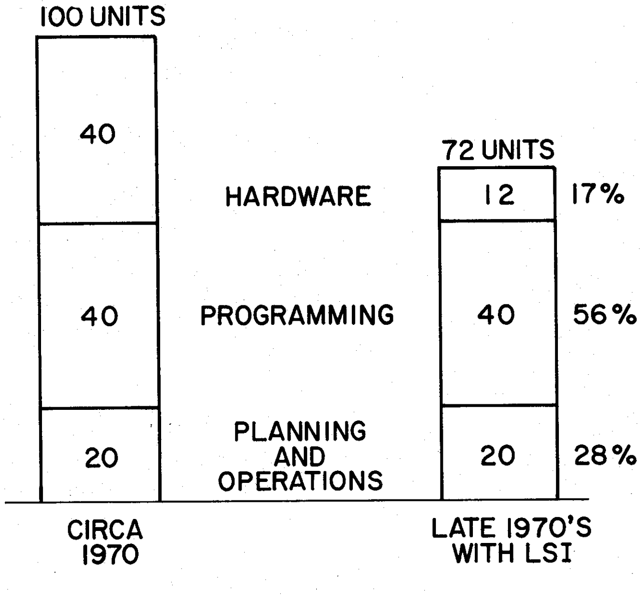


Figure 1—The usual costs of ownership

Interface capabilities—LSI gives us the opportunity to use low-cost logic and buffers for functional modularity and standardization. These features will improve ease of designing, installing, expanding, servicing and using systems, and will greatly increase flexibility. But, as always, there will be bottlenecks (e.g., response time) which can limit the number of functional interfaces—or require certain bypasses. However, providing a degree of universality to system components gives us the opportunity to build more components alike. This helps build up quantities per part number and aids in limiting the proliferation of development efforts. Such universality requires additional logic, because a part number has to be more versatile, but it is of benefit in reducing development costs.

Memory/logic clocking implications—One potential effect of LSI is the opportunity for design simplification afforded by a machine with equal memory and logic clock cycle times,[8] such as was the case in the days of tube computers. This possibility comes from using the faster bipolar memory (or a bipolar "cache" memory with additional slower memory), or smaller semiconductor memories which can be overlapped to gain speed with little penalty in cost. However, the access time is a higher percentage of the total cycle time than in core memories, leaving a smaller percentage for address generation or any logic operations on the accessed data.[9] To offset this requires relatively faster logic if the same functions are to be retained. An alternative is to limit the number of levels of logic, using higher memory speeds and simpler architecture. Either of these alternatives limits the economies of scale in going to higher performance machines. Of course, the option to not exploit the full memory speed-cost potential remains. The trade-offs are between speeds of logic and speeds of memory.

Interaction Among Costs, Task Management and Availability:

The Era of Multi-processors

1. *Costs*—At any technological point in time, there is an optimum processor size. The old rule that "bigger and faster" means "greater performance per dollar" does not hold over all combinations of conditions. In fact, at the high-power end of the spectrum today, cost approaches a linear relation with computing power.
2. *Task Management and Software*—Some features of programming systems can be simplified, if the speeds of processors and data storage devices can be brought closer together. If no more CPU resource is wasted by allowing the CPU to idle during a file read than would be consumed in overhead by task-switching, then it is preferable to let the processor idle—because of the potential reduction of time spent writing and debugging both operating systems and application programs. One way to approach this condition is to parallel-process with "slow" CPU's rather than single-process with a "fast" one. For example, a 1-million instructions per second (MIPS) processor executing 10,000 instructions per file access would be wasting only 10 percent of its time if it waited for a one millisecond average access time device, while a 5 MIPS processor idling during reading the same device would waste 50 percent of its time. In the past, we have multiprogrammed the "fast" CPU's to avoid waste because CPU costs were non-linear with speed. We have conserved CPU power by programming effort, but it may be time to reverse those factors, and multi-process instead.
3. *Availability*—Availability can be brought to any given level (excluding perfect!) by increasing the level of multi-processing.

Also, there will be workloads beyond the capacity of any processor that we will ever build; at this point, multiprocessing becomes a requirement, not an option. Of course, the processors might be packaged together under one cover; on the other hand, it might be prudent to separate them geographically to prevent non-hardware-caused reliability problems.

Software/firmware/hardware trade-offs—We hardly need to be reminded that micro-programming techniques will be important, and they are already widely accepted. LSI projections further enhance these techniques, particularly in writable control stores which offer greater flexibility. Many functions in teleprocessing systems today are handled by software, e.g., line control, bit handling, etc., but performance often warrants hardware solutions. Higher-level control programs and language programs in hardware or software become more realistic.[13] This will be the direction as the industry matures.

Extension of computer usage

There are a number of not necessarily independent ways in which LSI will extend the use of CPU's and memory:

- Broadening the user base in conventional areas.
- Extending small machine with large machine functions.
- Providing new and additional functions.
- Providing universality, standardization, flexibility.
- Meeting new performance criteria.
- Opening new application areas.

There is evidence of each of these today. Broadening the user base in conventional areas simply implies selling more systems to more users for established applications, usually at the small end of the line. More interesting is the addition of typically larger-machine functions to smaller machines,[4,5,8,10] e.g., more sophisticated instructions, memory protection, instruction retry. Such features could add an order of magnitude in logic and memory to today's small systems.

Adding function to existing systems is conceptually an extension of the aforementioned. For example, function will be directed at improving data acquisition and use, at extending the ability of a system to self-diagnose and reconfigure, at providing communications functions and security, or at converting current software functions. Clearly, this is an open-ended category and suggests the basis for many times present hardware levels.

Meeting new performance criteria, we believe, means placing higher priority on response time, availability, ease of installation and use—rather than on throughput. These objectives will require new approaches, and more logic and memory can profitably be put to use in their pursuit.

New application areas will be developed. However, there are many applications which are presently on the shelf because of either time or economic reasons. It is already apparent that environmental monitoring and control is an important area for development. Attempts at "duplicating" human function, e.g., constructing capable robots, will undoubtedly consume much circuitry. These today are analog functions and require two to three orders of magnitude more components when done digitally. Still, it is becoming more and more economical to digitize, and examples are well-known in the data transmission area. Other known areas such as pattern and speech recognition will consume extensive hardware when we exploit them to their fullest.

In the next section we will use a specific system, namely, a highly interactive data processing system, to *illustrate* the impact of LSI technology.

A HIGHLY INTERACTIVE TELEPROCESSING (TP) SYSTEM

One motivation for looking at this system is that we believe that ultimately there will be very large systems serving tens of thousands of users. The users will be indifferent and largely oblivious to the fact that they are using a computer system. They probably won't even call their terminals "terminals"; names such as "fact-finder," "stockinfo," and "smartmart" are more likely. In particular, these people will probably not be programmers, mathematicians, engineers, scientists, or accountants. Their applications will be logically trivial—at least from a systems person's point of view. What we today might expect to be a command language will be the jargon of their particular environment, and they will be largely innocent about how the data they reference was acquired or is organized.

The interesting features of the system are that (1) there will be a very high traffic rate, (2) the traffic will have to do mainly with references to or updates of a data base, and (3) response to an inquiry will have to be within no more than 1-2 seconds to satisfy the users psychologically. The terminals might be displays, keyboards, badge readers, etc.—with or without "intelligence." A high degree of reliability will be required, and we will not want to manually reprogram each such system for each environment. Airline reservations systems are a current example of such systems, but other systems of even larger size are destined for such areas as the financial, securities, and retail industries.

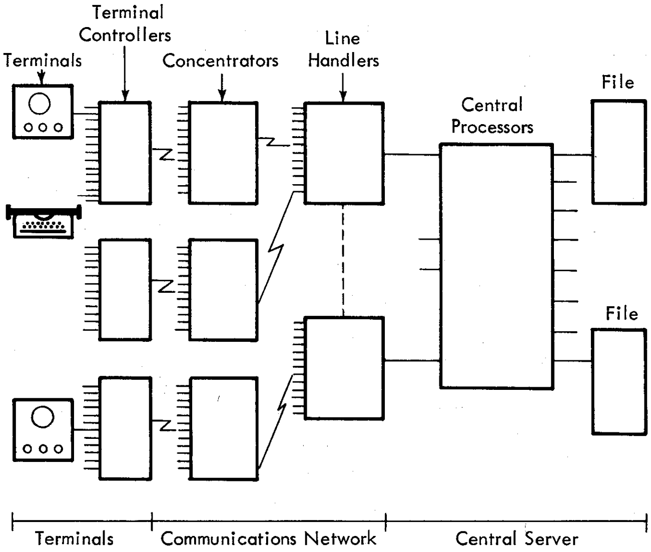


Figure 2—Schematic of interactive system

We postulate here an example of a hypothetical, highly interactive system for further discussion—in terms of potential LSI impact. The system is to have 10,000 terminals, and performance requirements are as follows:

Peak Message Rate	200 msgs/sec.
Simple Inquiry	30 percent
Simple Update	50 percent
Complex Inquiry and Update	20 percent
Data Base	10 billion bytes
Response Time (Simple Inquiry and Update)	1-2 sec.
Availability	1 failure/day with 2 min. recovery.

(In a real case, there should be additional qualifications on availability and recoverability.)

Some provision must be made for batch processing. Also, we have to look at data integrity, security, ease of installation, provision for growth and change, provision for on-line evaluation, etc.—each an area where hardware additions could contribute improvements.

Figure 2 illustrates the principal functions as we will describe them. Relating to current-day techniques, the peak load assigned to the central server can be projected to be the following:

Messages/sec.	200
File Accesses/sec.	1000
I/O Instructions/sec.	2.5×10^6
TP Access, Instructions/sec.	1.5×10^6
Instructions/sec. Total	5×10^6

For the purposes of this illustration, we assume that the configuration, in contemporary terms, uses four processors of the type described in Table II, plus two stand-by/batch processors (one might use fewer, but larger, processors). Using other component prices from reference sources[11] and our previous assumptions about changes in costs, we compute the cost comparison change shown in Table IV.

TABLE IV—A Hypothetical 10,000-Terminal Configuration

	*Cost Units** *Circa 1970*	*Cost Units** *Late '70's*	
Terminals (10,000)	59	20	60%
Lines and Modems	6	4	12
Multiplexors	1		
CPU Processor(s)	7	2	6
Channels, etc.	3		
Main Memory	4	.3	1
Bulk Memory	7		
Files	10	5	15
Other	3	2	6
	100 Units	~33 Units	100%

* Units: Equivalent to percentage of Circa 1970 costs.

Perhaps the most conspicuous feature of Table IV is the dominance of terminal costs. Secondly, perhaps, is the relative disappearance of the main and bulk memory costs. Next is the relatively small proportion of costs dedicated to transmission.

While the configuration assumed does not exist, it is at least realistic in the 1970 time frame, with some performance compromises. It is somewhat similar to an airlines reservation configuration in terms of the geographic assumptions made. The degree of terminal clustering achievable, relative to line requirements, makes it possible today to keep the percentage of transmission costs modest. However, it is assumed that line costs will not go down in cost as fast as the other components, and thus could constitute a higher percentage of costs—an opportunity for LSI to provide further concentration.

Even at the relatively low function level of 1200 logic gates per terminal, there is on the order of ten times as much cumulative logic at the terminals as at the central facility—and even more than 10 times as much memory if we use semiconductor storage at a display terminal for refresh memory. Clearly, it would be easy to increase these ratios significantly—if additional hardware at terminals could be utilized productively.

Just as important is the fact that with terminals dominating costs, and with logic and memory costs decreasing anyway, it will be less painful to add more function to the central facilities in support of systems which are more responsive and easier to install, use, and maintain. Of particular note here, again referring to Table IV, is the fact that the cost of a 20-fold increase in the size of main (and bulk) storage in the late 70's would still be possible within the indicated 1970 prices.

Figure 3 shows the response time breakdown that we would expect today, given a two-to-six second response time range and assuming relatively simple inquiry and update applications and moderate loading conditions. (The more conventional implementations would probably not achieve the one-to-two second response times we have postulated.)

The polling delay is half of the average polling cycle, initiated from the concentrator. The delays on the transmission lines are due to line speeds, message lengths, and queued waiting for the lines. Actual time delays in transmission are relatively negligible. Concentrator delays are principally queueing delays, with actual processing times in the 10 to 100 ms range. Central processing times are also mainly waiting times, particularly waits for file access, but also for channels and for processor service. Again, tens or hundreds of milliseconds are the range of actual processing times needed, subject, of course, to the processors' speed and the job to be processed.

It should be clear that these times are satisfactory for many applications; however, our contention is that a substantial segment of the potential user community will want response times in the one or two second range, or perhaps faster, at least for simple inquiry-update messages.

Design for the late 70's

Most systems used for interactive requirements today have been built upon batch systems. There have been attempts to achieve good throughput with acceptable response times. On this base, we are now adding more sophisticated user-oriented functions, in the area of data management, for example. In the future we must do more. The question is: "How far can we continue to build up the processing requirements on

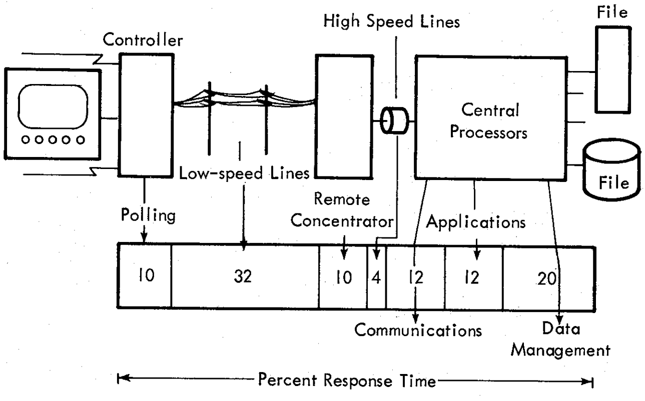


Figure 3—Factors in response time

the present base, and maintain, much less reduce, response times?"[11] We believe there are significant changes possible based upon the advances in LSI technology.

Present systems attempt to conserve processor resources. As we have said before, multiprogramming lets us use the processor while it would have been otherwise waiting on file accesses. Many-terminal systems, airlines for example, require us to maintain many messages active in the CPU concurrently, up to 50 in some systems. Caring for these messages is complex, and it is this type of complexity that causes much of the problems of installability, maintainability, etc.[12] In the future, processors, including memory, do not need to be conserved to the same degree.

LSI opens new possibilities which we could not consider in the past, particularly if we can appropriately partition the data bases (as we can in some environments today). We can bring the CPU speeds and random access speeds into better balance, we can afford to let the CPU idle during file reads and reduce the software and system complexity. The path to arriving at the balance appears to embrace (1) keeping the CPU suitably slow, (2) making direct access storage suitably fast, and (3) paralleling CPU's and direct access devices sufficiently to absorb traffic. In an environment where cost is nearly linear with CPU power, we can make the choice of reducing complexity at no cost, and into the bargain get increased reliability and perhaps improved security. Conceptually such a system might look like Figure 4.

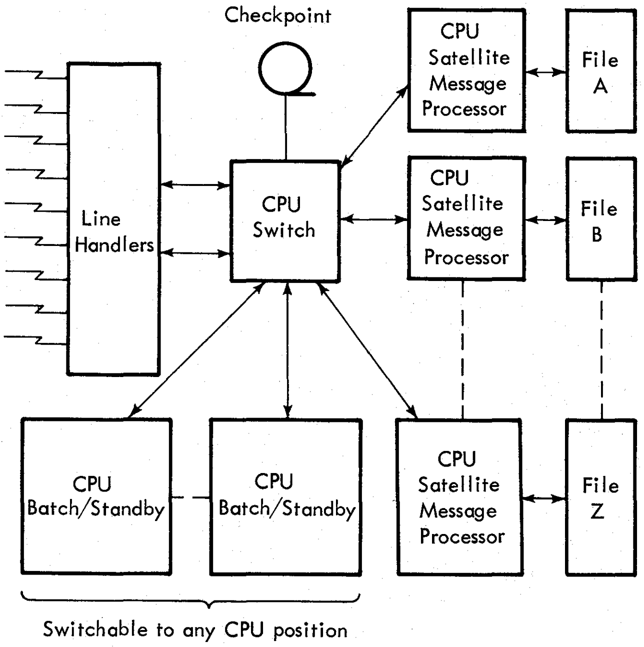

Checkpoint
CPU Satellite Message Processor
File A
Line Handlers
CPU Switch
CPU Satellite Message Processor
File B
CPU Batch/Standby
CPU Batch/Standby
CPU Satellite Message Processor
File Z
Switchable to any CPU position

Figure 4—A central server configuration for a highly interactive system

Communications processing

The many functions relating to the handling of input and output messages may give rise to separate communication processors. The precise role of these processors in the total teleprocessing picture, and particularly the interface to other central processing functions, has not been universally defined. In fact, the communication processor function should be redefined in the context of the total teleprocessing requirement, including response times, to assure that there is no unnecessary data handling, and that the communication processor does, in fact, efficiently complement the central processors. Here LSI will permit us to build the necessary buffers and other interface hardware to afford a clean interface between the communications functions and other processing requirements, thus reducing device dependence and improving interprocessor communication.

Line attachment is another area where LSI will help. This area includes the physical attachment, modem functions, switching, i.e., functionally getting the information on and off lines and into and out of registers. Interface standards (EIA) usually have to be met, a variety of line speeds accommodated, and, often, provisions must be made for using the dial telephone network.

At the line end of the communications processor, we have a choice between hardware and software. The 10,000 terminals, for example, will require many connections—and in the past this would have been very expensive. This would have been particularly true if we had implemented sophisticated equalization equipment as well as the other modem functions digitally. On the other hand, done strictly with software, these functions would require extensive processing and processing times. Fast response time requirements and LSI costs make hardware the clear choice.

LSI has also made another communication processor design choice easier. Here, a writable control store (WCS) approach gives the flexibility needed to exist with the many variations found in practice, i.e., terminal and line types, line disciplines, formats, etc. When communications processors must be fast, bipolar techniques can be used for the WCS—and perhaps for their main memories as well.

The communications network

Communicating between the central site(s) and the terminal in the many-terminal system can be a relatively expensive proposition but in most cases it need not be. In the example of Table IV, it will be about 12 percent of the cost. This is a function of the distribution of terminals—and since in most applications terminals tend to be clustered, there is the opportunity for the terminals to share fewer transmission lines. This enables us either to use lower speed communications lines more effectively or to afford to use a broader transmission channel, where the potential costs per bit transmitted are significantly lower.

Concentration of line sharing can take place in several places, including the central site, intermediate remote points, and multiple-terminal sites. Concentrators, since they amortize their costs over more than one terminal, provide us with the opportunity to make them far more sophisticated, and LSI will help us to add the optimum amount of function. Functions directed toward increased availability might take the form of off-site processing capability, alternate path routing, forward error correction and backup storage. For our interactive teleprocessing example, the response time requirement forces us to limit queues, particularly on slow lines. LSI can help here by enabling us to extend line capacity (through multilevel modem equalization, error correction, etc.), or to speed up communications processing. Again, we can under-utilize facilities more economically to combat queue build-up.

Figure 3 shows a plausible breakdown of delays for a 1970 configuration. (There are faster airline systems but they are configured somewhat differently and are otherwise specialized.) If we have a one-second response time objective, some reduction in nearly all of the time components must be realized, but the use of additional hardware for faster polling and message processing makes the overall one-second response time very reasonable.

Terminal functions

With the assumption that LSI will enable us to expand terminal function significantly, what will it mean for the future terminal? Clearly, the minimal function of the terminal is its I/O: keyboard, printer, display, etc.; the I/O control; the basic communication functions including code conversion, serial/parallel conversion, and parity checking. We can expand these functions by adding character, line, and message buffering; various modem functions; line control; security provisions. Further expansion could include local user prompting, message format editing, header generation, and a variety of more sophisticated error control and automatic retransmission features. Continuing further, the high availability features of error correction, diagnostic ability, even self-repair, could be considered. Pre-processing, such as syntax checking and a limited degree of local problem solving, is a natural extension—continuing upward to significant stand-alone power.

To arrive at some idea of the level of logic complexity addressed here, we list in Table V the rough number of logic circuits (gates) and storage units (cells) associated with the various listed functions. (The reader should realize that the numbers are highly variable, subject to the particular application and the performance required. In addition, the functional hardware requirements are not necessarily additive since there may or may not be sharing of hardware, and in the more complex functions the option to perform them by mixes of software and hardware is generally warranted.) Hardware counts are rounded to the nearest 100.

Two of the more dynamic ranges in Table V are those of "Forward Error Correction" and the "Modem" functions, which could include very sophisticated forward error correction techniques and automatic equalization. It is not difficult to envision the use of substantially more logic and memory in the terminal. For example, if we add 10,000 circuits and 100,000 bits of memory at the projected costs (and also increment some of the other costs), the terminal will still be cheaper than its 1970 forerunner.

LSI will afford us the opportunity to define and economically design many functional interfaces. These interfaces will enable us to design terminal controllers, terminal I/O devices, and stand-alone terminals with extensive functional modularity—and to do this without jeopardizing the low-cost single-terminal requirement. Likely interfaces include the line-modem interface, the modem-data link control (DLC) interface, the DLC-terminal function—and eventually, the terminal I/O device interfaces. By choosing the appropriate

TABLE V—Hardware Requirements of Typical Functions

Function	*Gates*	*Cells*
Basic I/O Control	100-200	<50
Display Character Generation (ROS)	<50	1000-8000
Display Refresh (CRT)	<50	1000-25,000
Basic Communications Functions	100-200	<50
Extended Line Control Functions	500-1000	<50
Forward Error Correction	400-20,000	4000-6000
"Modem" Function	200-10,000	100-10,000
Line and Message Buffering	<50	200 & up
Editing Functions	200-400	500-3000
Calculator Functions	100-500	500-3000
"Typical" Small Computer	2000-4000	16,000-64,000 plus files

sub-function hardware modules, the controller is tailored to suit the appropriate line, the number and mix of terminal I/O devices, and other general functional requirements. Even within the smallest hardware module a degree of generality must exist which supports the standard interfaces. However, as our cost example illustrates, we can afford a significant increase in logic or memory count before we have a substantial increase in overall terminal cost.

SUMMARY

We have looked into the future of projected LSI logic and memory costs. The implications to the industry are dramatic. Without additional applications, the cost of the hardware base is a fraction of the cost of today's base. There will be an increased search for new applications and a change of emphasis on performance factors. LSI will support an era of greater accommodation of the user's desires, as opposed to forcing user conformity to machine—and data processing will be extended to a much larger user base.

Some important design and architecture trends are motivated by the need to meet other than batch processing and throughput requirements—for example, response time, availability, installability, usability. LSI will aid in establishing needed interfaces for both user and manufacturer, support the evolution of multiple processors, accelerate the use of firmware, and add many new functions to systems, some converted from software. It will not significantly restrict design freedom if the part number generation costs are as low as projected.

While LSI alone has not solved all problems, it has shifted the emphasis, changed the scale, enabled us to do some things better. "Large" will become larger, costs will be lower—and exploiting our new environment will be the major challenge.

REFERENCES

1 *Large-scale integration perspectives*
IEEE Computer Group—Computer Elements Technical Committee, Computer Group News 2 #6 pp 24-32 1968

2 *Computer memories in the seventies*
Computer Elements Workshop Computer 14 #1 pp 58-59 1971

3 R NOYCE
Bob Noyce of Intel speaks out on the integrated circuit industry
EDN/EEE 16 #18 pp 28-32 1971

4 M G SMITH W A NOTZ
Large-scale integration from the user's point of view
AFIPS Fall Joint Computer Conference 31 pp 87-94 1966

5 M G SMITH W A NOTZ E SCHISCHA
Economic questions of systems implementation with large-scale integration
IEEE Transactions on Computers C-18 #8 pp 690-694 1969

6 M G SMITH
The terminal in the terminal-oriented system
Australian Computer Journal 2 #4 pp 160-165 1970

7 J E BRYDEN
Visual displays for computers
Computer Design 10 #10 pp 55-77 1971

8 R RICE
LSI and computer system architecture
Computer Design 9 #12 pp 57-63 1970

9 D L HOUSE R A HENZEL
Semiconductor memories and minicomputers
Computer 4 #2 pp 23-29 1971

10 W A NOTZ E SCHISCHA J L SMITH M G SMITH
LSI—The effect on systems design
Electronics 40 #4 pp 130-141 1967

11 *Auerbach computer technology reports*
Auerbach Information Inc 1970

12 J MARTIN
Design of real-time computer systems
Prentice-Hall Publishing Company 1967

13 R RICE W R SMITH
SYMBOL—A major departure from classic software dominated von Neumann computer systems
AFIPS Spring Joint Computer Conference 38 pp 575-587 1971

The rationale for logic from semiconductor memory

by W. H. DAVIDOW

Signetics Memory Systems
Sunnyvale, California

INTRODUCTION

Memory is the LSI component which we know best how to fabricate. Compared to other LSI (Large Scale Integrated) circuits, it is almost "standard", and because of the regular interconnection patterns which exist between memory cells, it can achieve very high component densities and is relatively inexpensive to make.

A product such as a 4096-bit Bipolar ROS (Read Only Store) has some rather remarkable characteristics. When used properly, it can:

1. Become the logical equivalent of about 500 gates.
2. Be packaged efficiently in small 16 and 24 pin packages.
3. Dissipate about 1 MW per equivalent gating function.
4. Be customized for about $1.50 per gating function.

These benefits can only be obtained by using new logic design techniques which have not been extensively used in the past. One of these techniques is microprogramming. A great deal has been written about this technique but it has not really found wic'es)read application.

In this paper we will provide a brief introduction to microprogramming and then show how this technique, used in conjunction with semiconductor memory, can greatly reduce the cost of the control portion of a digital system.

INTRODUCTION TO MICROPROGRAMMING

The idea of microprogramming is normally attributed to Wilkes.[1] His stated objective was to provide "a systematic alternative to the usual somewhat *ad hoc* procedure used for designing the control system of a digital computer."[2] Actually, many logic devices being implemented today are far more complex than the computers which are deriving benefits from microprogrammed implementation. Some experts today are forecasting that most complex logic systems of the future will use microprogrammed techniques for the same reasons that motivated computer designers to select this alternative.

In a general sense, any sequential logic device can be viewed as transferring information from one set of storage elements to another through a logic network. In the case of a computer, the execution of an instruction involves a sequence of information transfers from one register in the processor to another, some of which take place directly and some through an adder or other logical network. A machine or computer instruction is made up of a number of these transfers which Wilkes likened to a program and hence, he termed the individual steps microinstructions and the sequence of steps a microprogram.

The mechanism Wilkes suggested for implementing the system is shown in Figure 1. The microprogram was held in a read-only diode memory or control store which is shown as consisting of two matrices A and B. The outputs of matrix A were used to control the transfers between the various registers. The outputs from matrix B were passed through a delay and were then used as inputs to a decoding tree. These inputs to the decoding tree directed the timing pulses to the next selected data line in the control store. Information from the sign flip-flop of the accumulator was used to select alternate paths through matrix B and hence to change the next selected data line in the memory. This is represented by one of the lines from matrix A which branches before it enters matrix B. The signals coming from matrix A control arithmetic gates, register paths, etc. Wilkes viewed these as incremental logical operations and called them micro-operations. A collection of these operations performed at one time was a microinstruction.

Wilkes' example segmented the microprogrammed control into two parts—one for operating the gates

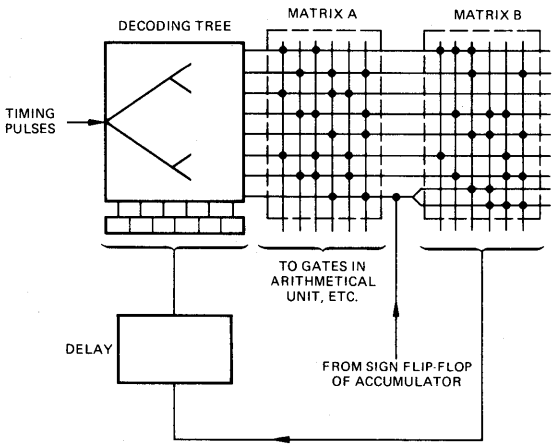

DECODING TREE
MATRIX A
MATRIX B
TIMING PULSES
TO GATES IN ARITHMETICAL UNIT, ETC.
DELAY
FROM SIGN FLIP-FLOP OF ACCUMULATOR

Figure 1—Wilkes' model of microprogrammed device

TABLE I—Microinstructions for simple machine

CODE	MEANING
MARM	Gate MAR onto M bus
MBRM	Gate MBR onto M bus
PCD	Gate PC onto D bus
IAD	Gate IA (instruction address) onto D bus
ACD	Gate AC onto D bus
R	Read memory into MBR
W	Write MBR into memory
PLUS	Add M bus to D bus in adder and place on S bus
DTS	Gate D bus through adder to S bus
MTS	Gate M bus through adder to S bus
ONE	Gate D bus through adder to S bus and add one
SMAR	Gate S bus into MAR
SMBR	Gate S bus into MBR
SPC	Gate S bus into PC
SIR	Gate S bus into IR
SAC	Gate S bus into AC
JMP	Jump to address in address field of microinstruction
JIFO	Jump to address in address field of microinstruction if AC = 0
JOPC	Jump to address of OP code field

which control the data paths, and the other for selecting the next step in the control sequence. Today all microprogrammed devices are organized in a similar fashion. The essence of designing a good microprogrammed system is the selection of the proper strategies for data paths and logic function control, as well as the implementation of the control sequence section.

Figure 2 shows a simplified computing element. In reality it could be a word-organized single-address stored program computer. In this over-simplified version, assume the control portion of the machine generates the signals shown in Table I. These signals are called micro-operations. By combining these, instructions can be generated. Assume further that the control portion of the machine contains a ROS (Read Only Store) and that words are read from the ROS under the control of an address register. Words are read from the ROS in sequence unless one of the Jump instructions (JMP, JIFO or JOPC) is executed.

Figure 3 shows an example of a ROS instruction word with the appropriate fields labeled. The symbolic representations of the micro-operations are shown under the fields in which they appear in the microinstruction word. There are of course many different microinstruction formats. For example, the Interdata Mod IV or Micro 810 computers use 16 bit formats, whereas systems such as the IBM 360/85 use 128 bit formats.

Figure 4 shows a microprogram in which the contents of memory are added to the accumulator in the following manner:

Line 1 The contents of the program counter are transferred to the memory address register.

Line 2 The instruction is read from memory into the memory buffer register.

Line 3 The instruction is transferred from the memory buffer register to the instruction register.

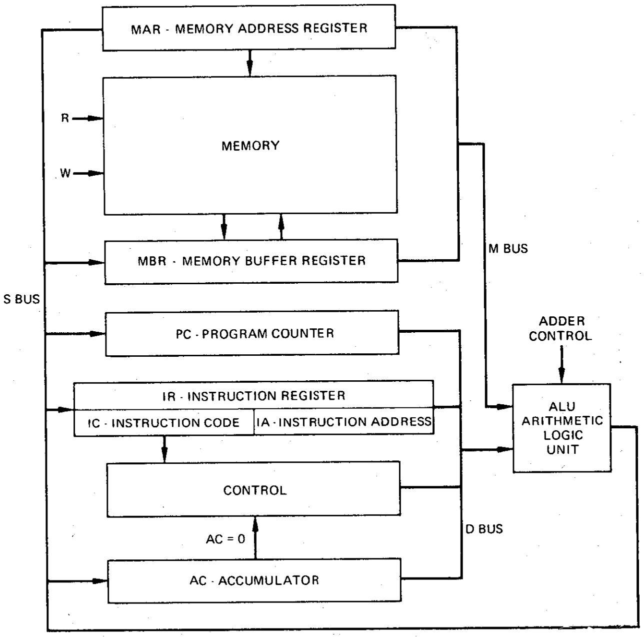


Figure 2—Simple microprogrammed machine

MEMORY CONTROL	JUMP INSTRUCTION	ADDER CONTROL	M BUS CONTROL	D BUS CONTROL	S BUS CONTROL	JUMP ADDRESS FIELD
R W	JMP JIFO JOPC	PLUS MTS ONE DTS	MARM MBRM	PCD IAD ACD	SMAR SMBR SPC SIR SAC	

Figure 3—Representation of a microinstruction word

Line 4 The control of the microprocessor is determined by the operation code of the instruction.

Line 5 The contents of the address portion of the instruction register are transferred to the memory address register.

Line 6 The desired data is read into the memory buffer register.

Line 7 The memory buffer register is added to the accumulator and the result stored in the accumulator.

Line 8 One is added to the program counter so it is ready to fetch the next instruction and the result is stored in the program counter.

Line 9 Control of the microprocessor is transferred to start the next sequence.

This example does not really discuss how the ROS is controlled. It merely points out that a control memory can emit signals capable of controlling the data paths in a system. The real essence of microprogramming is focused on controlling the address sequencing of the control memory itself.

CONTROL AND SEQUENCING OF THE CONTROL STORE

In every microprogrammed device there must be some logic which controls the ROS and does the basic system timing. The complexity of this logic depends to a high degree on the microprogrammed implementation. In a microprogrammed device with short ROS words where very few bits in the microprogrammed instruction word are devoted to determining the next instruction, the control logic will become fairly sophisticated. In these systems a ROS program counter will generally exist and the control logic will ensure that the next ROS control word is selected by augmenting the program counter by one just as the program counter is augmented in a typical computer system. In microprogrammed devices with longer words, the control will be conceptually simpler. For example, if each instruction contained the address of the next instruction, there would be no need for a program counter, only a ROS address register.

		MEMORY CONTROL	JUMP INSTRUCTION	ADDER CONTROL	M BUS CONTROL	D BUS CONTROL	S BUS CONTROL	JUMP ADDRESS FIELD
INSTRUCTION FETCH	1			DTS		PCD	SMAR	
	2	R						
	3			MTS	MBRM		SIR	
INSTRUCTION DECODE	4		JOPC					
EXECUTE	5			DTS		IAD	SMAR	
	6	R						
	7			PLUS	MBRM	ACD	SAC	
RETURN	8			ONE		PCD	SPC	
	9		JMP					START

Figure 4—Microprogram to add contents of memory to accumulator

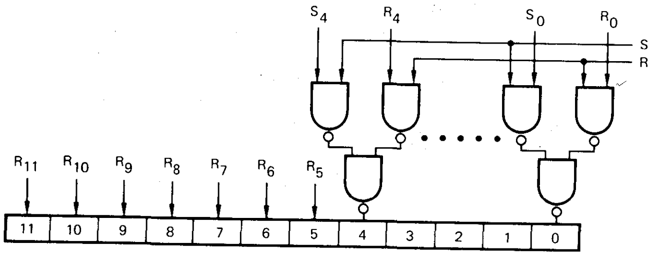


Figure 5—Two-address source for microprocessor address register

Wilkes' basic implementation shown in Figure 1 generated the address to read out the next microinstruction directly from the output of the B matrix. Therefore, if there were 1024 words of ROS, the output from this matrix would be a 10-bit code used to designate one of these words. While this technique is adequate, it fails to couple the microprocessing device in a tight fashion to its external environment and leads to an inordinately long response to an input stimulus. To see this, consider the problem of interpreting a 5-bit code and branching to one of thirty-two points determined by the code. This is the type of operation which occurs in interpreting the operation code of a computer. In the Wilkes' model, each bit must be tested separately and a branch must be taken at each step. This involves 5 separate tests, one for each bit and 31 branch-type instructions in a decision tree, plus an additional 32 branch instructions to get to the starting point of the various routines.

Wilkes' group suggested introducing multiple address sources. This greatly simplifies and speeds the solution to the above problem. To see this, consider the ROS address register shown in Figure 5 where the bits R_i are output from the address selection section of the ROS. Consider the bits S_j as bits from another register (for example, the instruction register in a computer). In order to interpret a 5-bit code, one need only let the bits R_{11} to R_5 select the starting address of a table in memory which contains 32 starting addresses of ROS sequences. The bits S_4 to S_0 could select any one of 32 entries in this table which could direct the microprocessing element to the proper location. Instruction decoding could, therefore, be accomplished with one microinstruction.

The importance of this technique goes far beyond the concept of instruction decoding. Any set of conditions could be introduced as an input in the low-order bit

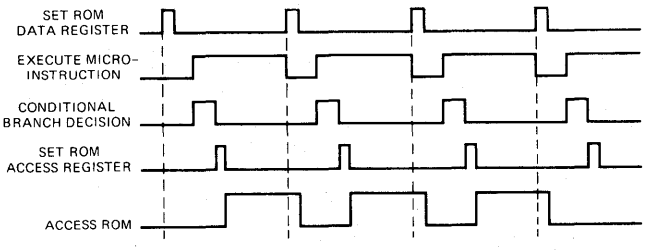


Figure 6—Typical ROS timing

positions and therefore, this represents one technique for developing a fast response based on the status of the external environment. For the example above, any one of 32 alternative courses of action could be taken in response to an input. This action could be initiated in one microinstruction cycle. With today's semiconductor memories, microprocessors with 100 nanoseconds cycle times are easily constructed, so that a response to an external stimulus can begin very quickly.

Figure 6 shows the timing in a typical microprogrammed system. The first set of timing pulses strobes the output from the ROS into the output buffer register.

As soon as the output buffer has settled, the execution of the microinstruction can begin. This is indicated in the second line of the timing diagram. The execution time will be long enough to permit the most basic microinstruction execution to take place. However, some of the longer operations might take two or more microinstruction times to execute. For example, suppose the time to gate data from source registers to the adder and to perform an "add" is 300 nanoseconds. If the basic microinstruction cycle is 150 nanoseconds, then an add operation would require two microinstruction times. This could be accomplished by executing the same microinstruction twice with the exception that data would not be stored in the destination register until the second instruction is executed. Another way to handle long instruction execution times is to let the microinstruction have a bit or bits which will be used to extend the microinstruction execution time. Still another obvious way for simpler systems is to merely let the longest instruction dominate the timing. For example, if 300 nanoseconds is the longest register-to-register cycle time, then the basic cycle could be selected as 300 nanoseconds. In most cases, however, a system will have certain microinstructions which require several basic machine cycles. For example, a fast microprogrammed device may be hooked to a slow core memory which has an access time of 800 nanoseconds compared with a basic microinstruction execution time of 200 nanoseconds. It is desirable to have some interlocks which enable this type of delay to be conveniently accommodated.

The fourth line in Figure 6 shows the time at which the ROS address register is loaded. This timing pulse will set the ROS address register. If a conditional branch is going to be executed, the logical decision must be made during the time shown in line 3 for the branch to be executed within the given cycle. Obviously, this is not always possible. For example, if a branch is to be made on overflow, it is likely that the information will not be developed by the adder until it is too late to enable a branch decision to be made for that instruction. Consequently, one instruction and one instruction execution time is lost because the branch decision must be made on the next instruction.

The fifth line of the timing diagram shows the time devoted to the access of the ROS. One reason for the importance of fast access in the ROS can now be understood. The shorter the access time, the more time is available for making decisions. Since a high percentage of microinstructions contain conditional branches, many instructions and instruction times can be saved by making the access time as small a percentage of the basic control cycle as is possible.

ADVANTAGES OF MICROPROGRAMMING

There are many reasons why it is desirable from a systems point of view to employ microprogramming. Some of these are:

1. Reduction of random logic, resulting in more structured organization.
2. Emulation of a number of devices (not just another processor) can be carried out by a single, general-purpose device.
3. Diagnostics and service aids may be implemented easily in the control portion of the system.
4. Field changes can be made easily by changing the contents of control stores.
5. Special-purpose adaptations can be made of a device by changing a few program cards.
6. The logical complexity of a device can be increased at lower cost than in a conventional system.
7. Final system definition can be postponed to a later time in the design cycle.
8. Documentation and service training costs can be reduced.

The prime motivation for interest in microprogramming today appears to be one of economy. To understand why, an analysis must be made of the cost of putting an IC (Integrated Circuit) into a system.

The direct cost of putting an IC into a system is over $1.00. Figure 7 shows a breakdown of these costs which one experienced manufacturer of digital systems considers reasonable. When all of the direct costs in fabricating a system are taken into account, such as indicator lamps, maintenance panels, system cabling, etc., and the total system manufacturing cost is divided by the number of IC's, one frequently arrives at a figure of $2.00 per IC. The $1.05 cost of Figure 7 will be used during the rest of this discussion because there seems to be general agreement that getting rid of one IC will usually save a manufacturer that much money.

It is interesting to note that under the assumption of Figure 7, even if IC's were free, the total manufacturing cost could be reduced by only 30 percent. Assuming designers continue to use conventional IC's and packaging approaches in design of their systems, there will be small, if not significant, decreases in the manufacturing cost of digital systems.

• IC	.30
• Incoming Inspection	.05
• P C Card	.25
• Component Insertion and Fabrication	.03
• Board Check Out	.05
• Connector	.05
• Capacitor	.03
• Wiring	.09
• Power Supply @ $1.00 / Watt	.10
• Cabinetry / Card Guides / Fans / etc.	.10
	$ 1.05

Figure 7—System cost for a 16 pin DIP on 88 pin 60 IC board

If one assumes that the average IC contains 3 gating functions, then the direct cost per gate installed in a system will be on the order of 35¢.

For the most elementary type of devices, random control logic will always be the most economical way to design a system. If, for example, a device can be controlled by a single pre-set counter, it is impractical to become involved in the intricacies of microprogrammed control. This is because a microprogrammed device has a base cost associated with the address sequencing and memory selection circuitry which is incurred no matter how simple the device is. This is shown in Figure 8. Properly designed semiconductor control storage has most of the memory selection circuitry integrated into the memory element and this greatly reduces the base cost.

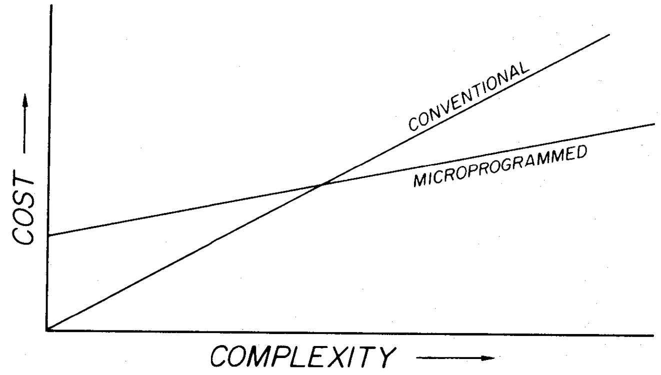


Figure 8—Cost comparison of conventional and microprogrammed control

Figure 8 shows the cost of microprogrammed control increasing slowly with the number of sequencing cycles. This is because each additional sequence cycle will, in general, require one additional word of control store. If, for example, the control memory of a device uses 32 bit words and the control store costs only 0.25¢/bit, as such devices will in the future, then the cost of an additional control cycle is only 8¢.

The cost of conventional control in Figure 8 is shown increasing at a fairly fast rate. One can see that if an additional control cycle required only one additional gating function or some type of storage element to distinguish it from all the other control states, then at least one gate or 35¢ in direct cost must be added to the system cost. For this simple example, the cost of conventional control increases at over four times the rate per sequencing cycle of microprogrammed control.

	1¢/BIT	0.5¢/BIT	0.25¢/BIT
• I C (4096 BIT ROM)	40.00	20.00	10.00
• Incoming Inspection	.50	.50	.50
• P C Card	.25	.25	.25
• Component Insertion and Fabrication	.03	.03	.03
• Board Check Out	.15	.15	.15
• Connector	.05	.05	.05
• Wiring	.12	.12	.12
• Power Supply $ 1.00 / Watt	.50	.50	.50
• Cabinetry / Card Guides / Fans / etc.	.10	.10	.10
	$ 41.70	$21.70	$11.70

Figure 9—Cost of ROM in microprogrammed system

There are a few hard facts to back up any estimates of the efficiency of microprogrammed devices, but most designers seem to agree that somewhere between 8 and 16 bits of ROM can be used to replace a gating function. If 16 is considered to be a pessimistic number and 8 an optimistic one, then a 4096-bit ROM is capable of replacing between 256 and 512 gating functions. This ratio provides a great incentive to the designer to employ microprogramming.

Figure 9 shows the direct cost of putting a 4096-bit ROM in a microprogrammed system, assuming costs per bit of 1¢, 0.5¢ and 0.25¢. If R bits of ROM replace a single gating function, then the dollars saved by employing a single 4096-bit ROM at 0.25¢/bit are:

$$\text{SAVINGS} = 0.35 \times \frac{(4096)}{\text{R}} - \$11.70$$

where $11.70 represents the cost of the ROM shown in Figure 9. These savings are plotted in Figure 10 for various costs of ROM.

While 12 bits is probably a realistic number per gating function replaced, it is interesting to note that the break-even points for 1¢, 0.5¢ and 0.25¢ per bit ROMS are 36, 66, and 122 bits. With a 4096-bit bipolar ROM there can be a tremendous margin of error in the assumptions, and still have microprogramming be the most economical approach to system design.

Other savings result as well. These are tabulated in Figure 11. For example, if one assumes 12 bits of ROM will replace a gate function, then the use of 4096-bit ROMs in the control portion of a digital system will save about 114 packages, 8.5 watts of power and 100 cubic inches of volume in addition to $119 per 4096-bit ROM employed.

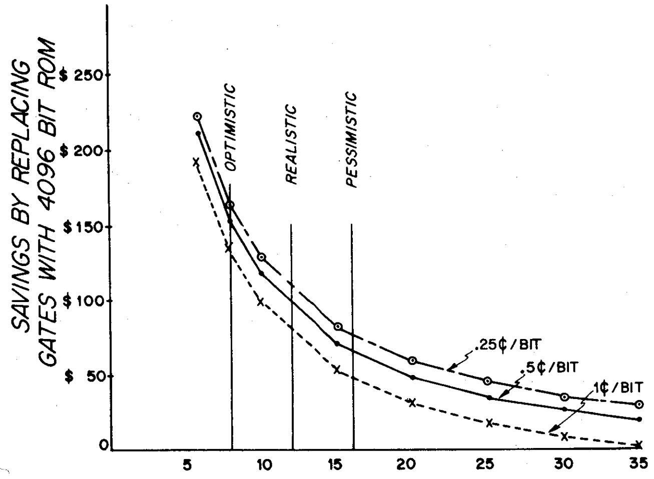


Figure 10—Number of ROM bits to replace a gate function

	BITS PER GATE FUNCTION		
	8 OPTIMISTIC	12 REALISTIC	16 PESSIMISTIC
NO. OF DIPS SAVED	170	114	85
PC CARDS SAVED 60 DIPS/BOARD	3	2	1.5
POWER SAVED IN WATTS	12.8	8.5	6.4
VOLUME SAVED IN IN³	150 IN³	100 IN³	75 IN³
DOLLARS SAVED PER 4096 BIT ROM	$168	$119	$83

Figure 11—Savings by using 0.25¢/bit ROM in microprogrammed control of digital system

CONCLUSION

The true promise of LSI has not been realized in most digital systems simply because it has been unappealing to use custom designed parts in all but the highest volume applications. Microprogramming techniques used in conjunction with semiconductor memory can provide many of the benefits of LSI for a broad class of systems. The designer, to take advantage of this new component, will, however, have to change many of his design techniques and philosophies.

REFERENCES

1 M V WILKES
The best way to design an automatic calculating machine
Manchester U Computer Inaugural Conference 1951 p 16+ff

2 M V WILKES
The growth of interest in microprogramming
Computing Surveys Vol 1 No 3 Sept 1969 p 139+ff

SYMBOL hardware debugging facilities

by M. A. CALHOUN

Kansas State University
Manhattan, Kansas

INTRODUCTION

The SYMBOL system is a full-scale working demonstration of an ALGOL-like, high-level, general-purpose, procedural language and a multi-processing, multiprogramming operating system implemented directly in hardware.[1,2,3,4] The results have proven the versatility of the design, construction, and testing techniques that were developed to enable a small number of people to implement such a major system in a reasonable time and at a relatively low cost.

Although the last items in the project to be developed, the methods used to test and debug SYMBOL were not the least important; delivery of the finished system would have been much delayed if more conventional methods of manual excitation and probing had been used. To facilitate fuller understanding of these testing techniques, a brief description of the architecture and physical characteristics of SYMBOL will be given first, followed by a more detailed description of the test facilities.

SYMBOL HARDWARE AND ARCHITECTURE

Logic in SYMBOL is implemented with Complementary Transistor Micro-Logic (CTμL), a positive AND/OR logic family with wired-OR capability at every output except that of the flip-flop.[5,6] The importance of the wired-OR capability must be emphasized: there are a significant number of two-way wired-OR ties with sources throughout the six-foot long main frame; if discrete OR gates had been required, the size of the finished system would have been considerably larger and its simple bus-oriented organization would not have been possible.

The dual-inline CTμL integrated circuits are assembled on about 110 large two-sided printed circuit boards with plated-through holes (see Figure 1). These boards contain all of the logic components in the system and are easily removed for incorporation of logic changes using discrete wire "patches" to the original printed circuitry. Figure 2 illustrates schematically how the boards are mounted between a pair of parallel two-sided printed circuit "motherboards".

Communication between boards is provided by two groups of 200 parallel printed circuit bus lines on the motherboards, the "connect" bus and the "bypass" bus. These buses provide all required system interconnections except for cables to peripheral devices and to the console. Physically, each large board can contact up to 200 bus lines and have the remaining 200 lines bypass it. A signal on the bypass bus may cross over a signal on the connect bus between board positions to allow a board to contact the corresponding position on the adjacent board or, alternatively, to a board several slots distant. In addition, either or both buses may have continuity breaks to isolate groups of boards.

There were eleven such isolated groups of boards in the original system, where each such Autonomous Functional Unit (AFU) acts as a specialized processor (see Figure 3). The Main Bus, a 111-wire subset of the 400 wires on the motherboards, runs the full length of the system and connects to each AFU. The lines in this time-shared, global communication path are distributed as follows:

64	bidirectional data lines
24	bidirectional address lines
10	bidirectional priority lines
6	operation code lines
5	terminal number lines
1	system clock
1	system reset

In order to fully appreciate the operation of the debugging facilities and the interactions between these facilities and SYMBOL, it is necessary first to under-

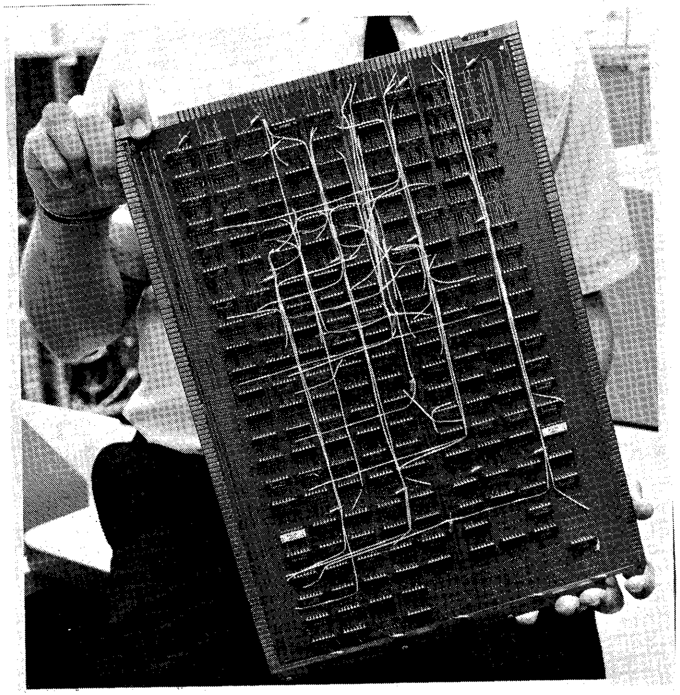


Figure 1—Typical 12″ × 17″ two-sided printed circuit board with logic additions and changes made with discrete wires

stand the various modes of communication which exist between the eleven AFU's. The foundation for this understanding is the knowledge that *everything* of importance that happens within an AFU eventually has some effect on the Main Bus. When the Instruction Sequencer finishes processing one instruction and needs another, a request to the Memory Controller is sent via the Main Bus. If the Arithmetic Processor runs out of data, a call for more goes to the Memory Controller. In both cases, information returned from the Memory Controller is sent over the Main Bus. If an input device completes its operation, the Channel Controller notifies the Interface Processor via the Main Bus. When the Translator is done with compilation of a program, the System Supervisor is notified via the Main Bus. Even if a processor fails catastrophically and refuses to function at all, the absence of Main Bus transfers originating from that AFU over a short period of time (20 clock cycles, perhaps) can be detected. Thus it can be said that whatever controls or watches the Main Bus thereby controls or watches SYMBOL. This concept of the Main Bus and the kinds of information that are transmitted through it are key features in the design of the total system and in the functioning of the debugging facilities.

Four distinct types of bus usage are possible. They are:

AFU to Memory Controller requests
Memory Controller to AFU returns
AFU to AFU transfers
Control exchange cycles

All Main Bus transfers are priority sequenced, where the priority bus coordinates the usage of the other buses. When an AFU desires to use the Main Bus, it raises it priority line and simultaneously checks to see if there are any higher priority requests pending; if there are none, it uses the bus on the following cycle. AFU to AFU cycles, used only between sections of the Central Processor, are also controlled by the priority bus: the Central Processor assumes the Main Bus is available for a private "sneak" cycle if no priority lines are raised. Control exchange cycles are used to communicate control information between the System Supervisor and the other AFU's. During a control cycle, the Main Bus lines have preassigned uses, with certain lines used to start and stop each AFU, others to indicate the completion of an AFU task, etc.; during a given

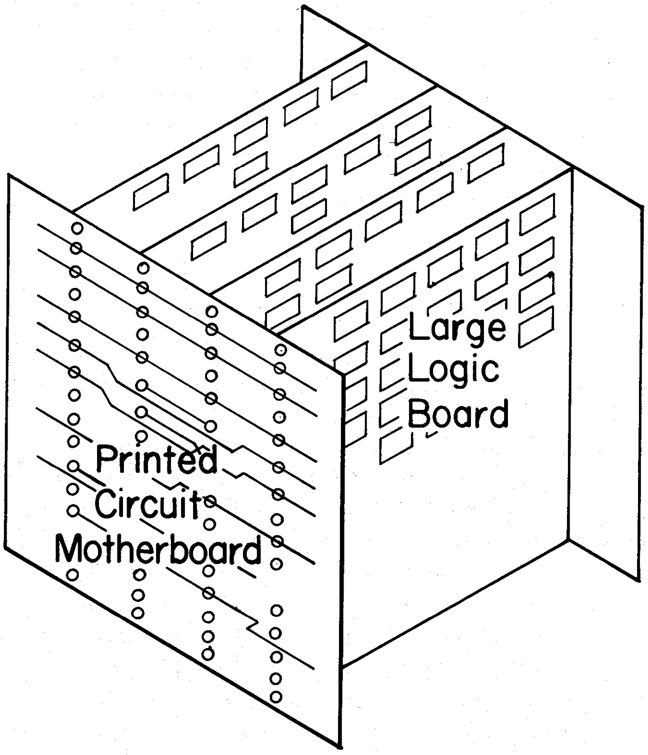


Figure 2—Physical organization of SYMBOL illustrating placement of large logic boards between parallel printed circuit motherboards

cycle, any combination of the paths can be used simultaneously.

SYMBOL DEBUGGING

The testing operations required in the development of a large system can be divided naturally into several fairly specific stages. The test functions vary as the system progresses from the research and development stage, to the prototype stage, to the production stage, and finally to the customer support stage. The emphasis on testing during these stages shifts from one in which strong support is given to engineering design checkout, to one which gives strong support to bad-part identification. System development within each stage requires component, board, AFU, and system testing functions; these functions are not always clearly distinct, but they can be considered separately for convenience. The discussion which follows illustrates the major requirements for testing in a research and development environment, with other aspects of testing discussed to a lesser extent.

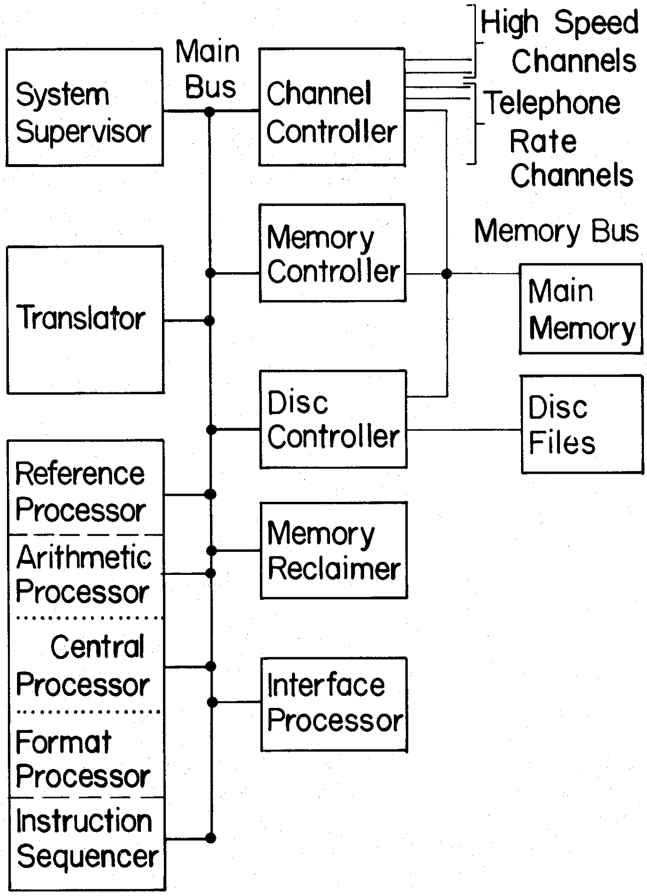


Figure 3—Block diagram of the SYMBOL system showing the original eleven autonomous functional units all connecting to the main bus

Component testing

Component testing is normally an incoming inspection operation. The integrated circuit devices used in SYMBOL were checked both statically and dynamically upon receipt for two main reasons: (1) The devices often had a long lead time for procurement, and an incoming inspection allowed immediate replacement orders to be made for bad devices. (2) The environment in which the devices were used (printed circuit boards) is one which has a high time (and therefore cost) penalty for replacing a device that should not have been installed in the first place.

The tests included a static check of the logic the device was supposed to implement, the on/off delay times from threshold to threshold, a check on the ringing tendencies, operation of the device when driven by a minimum clock pulse (where applicable), and the operation of the device under minimum and maximum power supply limits. These tests were made on a breadboarded device tester which, while adequate, was not very efficient, especially from a human factors viewpoint. In particular, the tester was capable of making only one class of measurements at a time; if a device required two classes of measurements, it had to be retested in a second pass. This resulted in a low throughput rate since about half of the time required to test a device was spent in manually transferring it from the source bin to the tester and from the tester to the "accept" or "reject" bin. Fortunately, better component testers are now available with good human factors design and automatic device transporters for quick and efficient incoming inspection.

Except for perhaps 100 devices received in the last few weeks of the project, all 18,000 integrated circuits in SYMBOL were given a thorough incoming inspection. It is safe to say that these last 100 devices caused more trouble during debugging than the first 18,000. This was sufficient proof of the absolute necessity for complete incoming component testing for future projects.

Board testing

Two major kinds of errors should be detected and corrected through board testing before a board is placed into a system. As in most system development, construction errors of various kinds will be discovered.

These errors will include fabrication errors such as drawing mistakes, missing or misplaced holes in the printed circuit, solder bridges, and misplaced components, to name just a few. In addition, there will also be system design errors ("oversights"?) which must be found and corrected. The scope of this problem was extremely large for SYMBOL boards because of their great number (110), their size (maximum of 220 integrated circuits), and their limited number of signal contact points (250). Considerable development is still necessary before automatic computer test generation is economically feasible for this size of board: current programs might require two hours to generate complete tests for about one-fifth of the logic on a board, and the difficulty of the task increases considerably with increasing size. Such costs could not be justified for this prototype system. If the problem were altered from an exhaustive test of all of the logic on a board to a test of all the used states of the logic, it might very well prove economically feasible. In particular, in a production environment, the cost of such test generation could be amortized over many systems since the same test sequence would be used over and over again. However, even though the isolation and correction of fabrication errors before system checkout began would have allowed more efficient debugging, it was felt that the amount of effort required for a thorough precheck could not be justified for this one-of-a-kind system. Thus, after a relatively cursory examination of each board for obvious errors (such as solder bridges) by experienced technical help, the board was returned to the designer for simple logic verification in a manual test station before proceeding to AFU or system test. This board test station has been described elsewhere.[4]

Subsystem and system test

Subsystem testing was the process of debugging a complete Autonomous Functional Unit as a unit. Each AFU performs, by itself, some function in the complete system, and, in a sense, is a stand-alone unit. Except for the Memory Reclaimer, a two-board AFU which was extensively (but not completely, as it turned out) simulated using the FAIRSIM* system, most AFU's were too large to allow simulation as a unit; therefore, subsystem test was the first time at which all of the boards in an AFU were run together. By this time most of the fabrication errors had been detected through board testing, although there were always a few left to find.

Subsystem testing could take many forms and could be performed in many different ways. An extension of the board test station was conceived where all boards of an AFU could be inserted into one module and a manual test performed. This sort of testing might have been useful in some limited areas such as verifying the basic cycling of control logic or the basic register transfers within an AFU, but, in general, a complete AFU test in this sort of test station would be very difficult, inefficient, and time consuming. Recognizing, then, that a thorough job of AFU testing was not desirable or even practical, subsystem and system testing were combined into the same operation and were essentially carried on simultaneously.

System testing was quite a prolonged process and involved a wide spectrum of activities from the initial startup of each of the AFU's on through to the final operation of the full system. The debugging of SYMBOL encountered many unique problems analogous to trying to test both a central processing unit and an operating system simultaneously on a more conventional computer. Unfortunately, self-diagnostics were impractical until late in the project because the construction schedule was such that Input/Output equipment was one of the last items to become operational. Some way of automatically exercising and/or monitoring various AFU's and portions of the system and of recording the results of these tests was absolutely necessary; in response to these needs, the first System Tester was developed.

SYSTEM TESTER

The System Tester consisted of three major elements: a programmable minicomputer, a logic interface and bus multiplexer between the minicomputer and SYMBOL, and a comprehensive set of computer programs which could control (and be controlled by) events

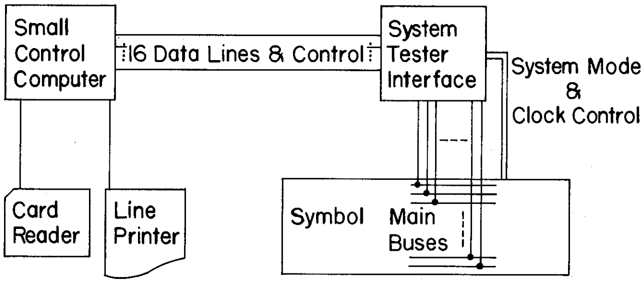


Figure 4—Block diagram illustrating connection of the programmable control computer to SYMBOL through the system tester interface

* Fairchild Digital Simulation program for Computer Aided Design.

occurring on the Main Bus. The configuration is schematically represented in Figure 4.

The minicomputer was an IBM 1130 with an 8192 word, 16 bits per word, memory (3.6 microsecond cycle time), a removable disc, a console keyboard and typewriter, a card reader/punch, a Data Products 80-column high-speed line printer, and a Storage Access Channel. The standard software supplied with the system (Disk Monitor Version II, Level 3) was used throughout the project.

The hardware interface consisted of four large logic boards between a pair of short motherboards mounted in a small separate enclosure (along with the necessary power supplies, cooling fans, etc.), placed adjacent to one end of the SYMBOL main frame. Short cables were used to connect the Main Bus to the interface, and a somewhat longer and thinner cable connected the interface to the control computer. The interface to the line printer was located on a fifth board in this same module.

The System Tester interface logic was divided into four basic sections. The first was a fairly small one which consisted of a number of TTμL inverters which were the logic level buffers between the control computer and the CTμL logic in the rest of the system. Second was a set of control interface registers and Input/Output logic which communicated with the control computer. The third section was another set of data registers which could be loaded and unloaded by commands from the control computer and which were used to transfer data to and from the Main Bus. The last section was the control logic which obeyed the priority rules of the Main Bus and included system reset and clock controls.

In addition to the 111 Main Bus lines, four extra lines were allocated to a System Test Mode which was distributed throughout the system and decoded by certain AFU's to provide additional control of several of the more complex operations. Different test modes, for example, could completely turn off a particular AFU or could force serial instead of parallel operation of all the AFU's. The discrepancy between the speed of the control computer's 16-bit data channel and the considerably larger Main Bus was resolved by allowing the interface to turn off the SYMBOL clock momentarily each time data was transferred to or from the interface buffer registers. Although control of the clock in this manner significantly lowered the execution speed of SYMBOL, all logic sequences were still executed in the same order with respect to each other, except for real-time peripheral devices, no matter what the actual clock speed.

The third major element of the System Tester was a language called DEBUG (what else!) and a set of programs which interpretively executed this language to control the interface as directed by input from the user; details of the language are given below. The actual interface and interrupt-servicing subroutines were written in assembly language while the majority of the remaining programs were written in FORTRAN. However, as new features were added to the system during the development period, the limited memory of the control computer was exhausted several times; each time, another routine was converted from FORTRAN to assembly language in order to free additional space. Notwithstanding this severe memory limitation, only rarely was a language feature removed, and then only after assurances from all users that the feature really was not being used.

In addition to the specific language features described below, many specialized programs were written (normally in FORTRAN) to provide formatted memory dumps, save-restore capability between the removable disc and SYMBOL memory, and other convenient features.

DEBUG LANGUAGE

The following sections describe the DEBUG language and the operation of the System Tester in some detail and especially attempt to convey some of the flavor of the system. The past tense is used because the System Tester was replaced by a special-purpose AFU, the Maintenance Unit (to be described later) and thus no longer exists.

The interpretive DEBUG processor accepted commands from punched cards. Each command consisted of an instruction element followed by a list of zero or more data elements. The format was relatively free form, with all blanks (spaces) ignored except in a string data element. Each element was separated from the preceding one by a comma or an equal sign. The last element was terminated by a period. Comments or other information following the period were ignored (a comment card was one whose first non-blank character was a period). Blank cards could be included in the deck for readability.

Extensive error checking was performed by the DEBUG processor, although very little error recovery was attempted because of the paucity of memory. When a mistake was discovered, the main response was an indication on the output listing of its nature. The system's action after an error was not predicted. In addition to a listing of the input commands and error messages, other messages of an informative nature and

TABLE I—List and Description of Programmed Register and Interface Hardware Register Mnemonics Recognized by the DEBUG Processor

Programmed Register Mnemonics

COMB—Address last formed by a COMBINE command
COUNT—Number of Main Bus transfers monitored in tracing
SPLTG—Group-link word address from a SPLIT command
SPLTP—Page address from a SPLIT command
SPLTW—Word address from a SPLIT command
ZERO—Infinite source of *zeros*

Interface Hardware Register Mnemonics

MP—Monitored SYMBOL priority bus lines
OP—Memory operation code last transmitted or error code last received
PL—Page-list currently specified (the mnemonic PAGE LIST could also be used)
PRIOR—Priority of the System Tester (any of the AFU's could be simulated)
RADR—Address last returned from SYMBOL
RCVX—Patchable register last received from SYMBOL
RDATA—Data last returned from SYMBOL
TN—Terminal number currently in use (TERMINAL NUMBER could also be used)
XADR—Address last transmitted to SYMBOL
XDATA—Data last transmitted to SYMBOL
XMTX—Patchable register last transmitted

the results of any action taken could be obtained on the line printer.

Data elements

A data element could have one of the following forms:

1. A hexadecimal constant formed by the character "/" followed by zero or more hexadecimal digits. Only the last 16 were retained. A typical example is /0079 004F C0 02D63B, where blanks have been included for readability. If fewer than 16 digits were entered, the result was right-justified. For left-justification ("contraction" is a good mnemonic) the last digit could be followed by an apostrophe and a character to designate the "fill" digit. Examples are /FF'O, where the leftmost byte is all *ones* and the rest are *zeros*, and /'A, a word of alternating *ones* and *zeros*.
2. A positive decimal number formed by zero or more decimal digits. Two examples are 0 (zero) and 18 446 744 073 709 551 615, the largest possible 64-bit number.
3. An alphanumeric string formed by a string-start indicator (an apostrophe) followed by zero to eight characters. To allow for character codes not represented on a keypunch, triplets composed of an escape character (the @) followed by a two-hex-digit coding of the desired character could be included in the alphanumeric string.
4. A data element could also be a register mnemonic formed by a single letter "A" through "Z" or a word chosen from a list of programmed registers or interface hardware registers which had specific predefined uses. The single-letter programmed registers were for temporary storage of data, addresses, etc., as the user desired. They could be initialized, cleared, etc., only by DEBUG commands or by being explicitly named in a Memory Control or System Supervisor command. A list and description of the multi-letter register mnemonics is given in Table I. Programmed registers A through Z and ZERO and hardware registers XDATA and RDATA were eight-byte (64 bit) registers. To refer to fewer than eight bytes, the mnemonics could be followed by a subscript from 0 through 7 enclosed in parentheses. Subscripts 3, 4, 6, and 7 implicitly referenced a single-byte datum. Subscripts 0 and 2 referred to a two-byte page address, and subscripts 1 and 5 referenced three-byte word addresses. No subscript implied context-dependent use of some or all of the full 64-bit register.
5. And finally, a data element could be empty or null if it were formed by two consecutive terminators or two terminators separated only by blanks.

The System Tester was a combination hardware/software system. To execute each command, the DEBUG processor transferred the contents of the indicated programmed registers to the appropriate hardware registers in the interface and initiated the proper operation. Although a data element could be left unspecified, the corresponding hardware interface register still participated in the operation; its contents were simply the value remaining from the previous operation. These registers could be initialized, set, cleared, etc., by DEBUG commands or by implicit side effects of various Memory Controller or System Supervisor operations.

Instruction elements

An instruction element was a mnemonic chosen from a predefined list; only the first five characters of mnemonics containing more than five characters were used

TABLE II—List of All DEBUG Main Bus Transfer Commands with Data Elements Explicitly Identified by the Corresponding Default Hardware Interface Register Mnemonic

Command	Function	Side Effects
AG, XADR, RADR.	Assign group	RDATA=0
AG00, RADR.		XADR=RDATA=0=PL
AG01, RADR.		XADR=RDATA=0
AG10, XADR, RADR.		RDATA=0=PL
DE, XADR.	Delete end of group	RADR=RDATA=0
DS, XADR.	Delete string	RADR=RDATA=0
DL, XADR.	Delete page list	RADR=RDATA=0
FD, XADR, RADR, RDATA.	Fetch direct	
FF, XADR, RADR, RDATA.	Fetch and follow	
FL, XADR, RADR, RDATA.	Follow and fetch	
FR, XADR, RADR, RDATA.	Reverse follow and fetch	
IG, XADR, RADR.	Insert Group	RDATA=0
RG, XADR, RDATA.	Reclaim group	RADR=0
RG0, RDATA.		XADR=RADR=0
SA, XADR, RADR, XDATA.	Store and assign	RDATA=0
SD, XADR, RADR, XDATA.	Store direct	RDATA=0
SI, XADR, RADR, XDATA.	Store and insert	RDATA=0
SO, XADR, RADR, XDATA.	Store only	RDATA=0
SS, XDATA, XADR, OP, RDATA.	Control exchange cycle	

for recognition. Most commands were order dependent, and data elements had to be listed in the exact order shown (or not listed at all). Data elements to the right of the last one of interest to the user did not need to be indicated. An example of each Main Bus command, with data elements explicitly identified by the default hardware interface register mnemonic, is given in Table II.

In addition to the simple Main Bus commands, there were 38 other commands available to the user to provide for explicit control of the System Tester environment; control of the hardware interface registers; maintenance of the files stored on the control computer's disc; loading, reading, writing, and testing of SYMBOL's memory; monitoring of SYMBOL; and decision making within DEBUG. These commands are summarized in Table III, where reserved mnemonics are shown in capital letters, and parameters and explanatory notes are given in lower case letters.

MAINTENANCE UNIT

To provide an independent maintenance and debugging capability, the System Tester was replaced by another AFU, the Maintenance Unit. Figure 5 schematically illustrates the final system configuration as delivered to Iowa State University. Input/Output equipment, controlled by a logic switch, is shared between the Maintenance Unit and a high-speed batch

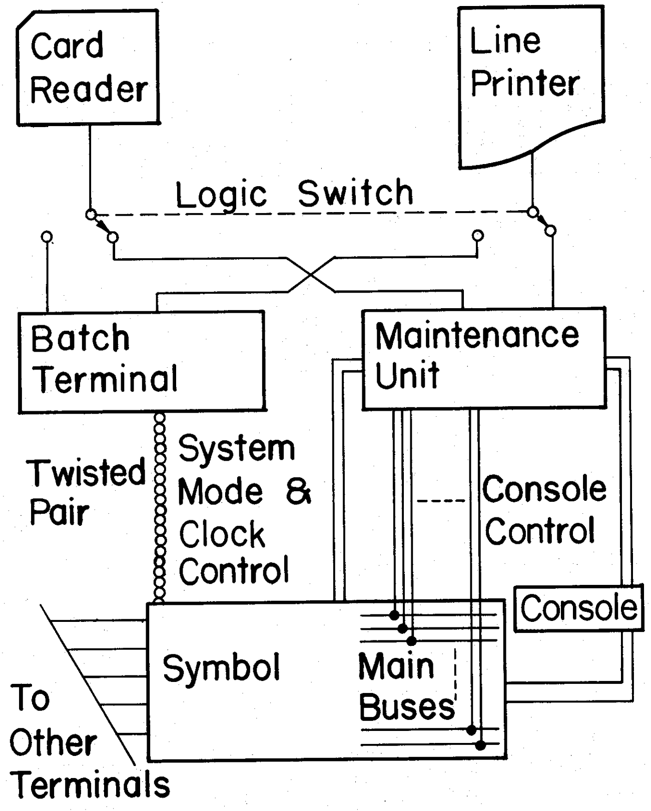


Figure 5—Schematic representation of the final system including the maintenance unit and a high-speed terminal sharing common input/output peripherals

TABLE III—DEBUG Macros and Commands for Controlling the System Tester

Control of the general test environment	
CLEAR TO ZERO, unordered list of registers.	
COMBINE, page number, group number, word number, COMB.	Address generator
EJECT.	Skipped output listing to a new page
END.	Signaled end of a testing session
EQUATE, old mnemonic = new mnemonic.	Corrected simple keypunch errors
HELLO, user's identification.	Identified beginning of a test session
LIST ALL.	Permitted printing of everything
LIST ERRORS ONLY.	Only error messages were printed
MOVE, destination = source.	Register-to-register transfers
PAUSE, number of milliseconds.	A timed delay.
PRINT, unordered list of registers.	
REGISTER STATUS.	Printed contents of interface registers
REPEAT LAST MEMORY COMMAND, number of times.	With XADR = RADR each time
SET, register = value.	Register initialization
SPLIT, source, SPLTP, SPLTG, SPLTW.	Address decomposer
STOP.	Press "START" on console to continue
WATCH, unordered list of hardware interface register mnemonics.	
Control of the hardware interface	
LOAD BUS AND STEP CLOCK, XDATA, XADR, OP.	Single-step bus control
START CLOCK.	Let SYMBOL free-run
STEP CLOCK AND PRINT CONTENTS OF MAIN BUS, number of times.	
STOP CLOCK.	Force SYMBOL to halt
SYSOFF.	Ccntraction of "system off"
SYSON.	Contraction of "system on"
Maintenance of files stored on control computer's disc	
DELETE, FILE n.	Deleted a previously saved file
FILE STATUS.	Gave status of all files on disc
RESTORE, FILE n.	Loaded SYMBOL memory and DEBUG status
SAVE, FILE n.	Stored SYMBOL memory and DEBUG status
Loading, reading, writing, and testing of SYMBOL memory	
FIND, beginning address, ending address, data, mask.	Searched for data
INITIALIZE, first address, last address, datum.	Loaded memory with datum
LIST, beginning address, ending address.	Dumped memory on line printer
LOAD, beginning address, ending address, ordered list of data elements.	
TEST CORE, XDATA, number of times.	High-speed repetive memory test
ZERO CORE.	Cleared SYMBOL memory to all *zeros*
Monitoring of SYMBOL	
PREPARE TO TRACE, unordered list of AFU names.	Set up for trace mode.
TRACE, XDATA, XADR, OP, unordered list of conditions for exiting.	
Decision making within DEBUG	
GO TO, label.	Skipped cards until label was found
IF, data, mask, condition, skip flag, number of cards.	Decision maker
LABEL, user-chosen label.	Provided target for a GO TO command

terminal. The Maintenance Unit console, with START, STOP, INHIBIT LIST, and RESET switches, is integrated into the SYMBOL console. A simple punched card input, easily produced at a keypunch or through another processor called SUPERBUG (written in FORTRAN for an IBM System /360) is used to direct the hardwired Maintenance Unit. To preserve the results of all prior testing, all DEBUG card decks were converted to equivalent SUPERBUG decks through a modified version of DEBUG which used the System Tester and SYMBOL in the translation process. The essential features of this control language are given below in hopes that they will provoke additional ideas.

The Maintenance Unit is a card-driven character processor where each card contains one or more commands. The command format is space independent, but parameters are order dependent. The first non-blank character on a card is the command code. Parameters (if any) are separated from each other by a dash (minus sign). Two consecutive dashes (with zero or more intervening spaces) indicate an (optional) missing parameter, in which case the register contents remaining from the previous command are used. The first semicolon on a card terminates processing of further characters from the card (i.e., human-readable comments, sequence numbers, etc., may follow a semicolon) and the command is executed with parameters received before the semicolon (a semicolon is automatically generated at the end of a card if none were previously found). Because of minimized decoding logic, the only valid characters before a semicolon are the hex digits 0 through F, the dash, and the (ignored) space.

Maintenance unit hardware

Table IV describes the hardware registers contained in the Maintenance Unit. In the table and in the following descriptions, each upper case character in a register mnemonic represents a single card column.

During operation, XXXXXX and Q are cleared to *zeros* before each new command is read. RRRRRR, RC, and DDDDDDDDDDDDDDDD are preserved from one instruction to another unless changed by a memory command. M is preserved until a new value is loaded or until a system reset is performed, after which M=0. XMP, CCCCCC, Ø, P, and TN are preserved from command to command unless specifically changed.

TABLE IV—Description of Registers in the Maintenance Unit

CCCCCC—	Address compared during trace (24 bits)
DDDDDDDDDDDDDDDD—	Transmitted or received data (64 bits)
M—	System test mode (4 bits)
Ø—	Operation code (4 bits)
P—	Page list (2 bits)
Q—	Disc quadrant code (4 bits)
RC—	Memory Controller return code (6 bits)
RMP—	Priority seen during trace (12 bits)
RRRRRR—	Returned address (24 bits)
TN—	Terminal number (5 bits)
XMP—	Priority watched for trace (12 bits)
XXXXXX—	Transmitted address (24 bits)

Note that P, TN, Ø, Q, XMP, and CCCCCC are treated as a single ordered parameter when loaded from a card. P and TN may be changed without affecting Ø by following TN with a dash or semicolon (or by leaving the remainder of the card blank). Other than these registers, the Maintenance Unit contains no memory; card commands are processed "on the fly" without benefit of additional storage.

SUPERBUG commands

The thirteen valid operation codes are summarized in Table V, where typical cards and variations are shown. Explanatory notes are again written in lower case letters. Unused command codes 0, 1, 2, 6, and - are simply ignored.

DEBUGGING TECHNIQUES

Although the command and control languages for the System Tester and the Maintenance Unit differed somewhat (with the Maintenance Unit having a considerably smaller repertoire), debugging using the two systems was essentially the same. Both systems allowed the full range of Main Bus transfers to be initiated. This meant, for example, that the Memory Controller (the central hub for most information transfers) could be thoroughly and exhaustively checked without having any other AFU functioning. Control exchange cycles enabled the starting and stopping of any AFU or groups of AFU's. The environmental commands and the macros, such as ZERO CORE, provided a good human interface with the system and permitted easy setup of initial conditions in preparation for a test. And last, and definitely *most* important, was the ability of both testers to monitor or "trace" the operation of SYMBOL as it operated under its own steam.

In the System Tester, monitoring was accomplished by a PREPARE TO TRACE card (which set the interface into a trace mode) followed by a TRACE command with parameters to start the proper AFU(s) and to establish stopping conditions. A single card combining both setup and startup commands was used with the Maintenance Unit. In either case, once the monitoring process was begun, all (or any subset) of the events occurring on the Main Bus (plus several "patchable" lines used to assist in observing internal AFU operations) could be printed on the high-speed line printer. At the option of the user, printing could be suppressed and the results of the trace could be stored "round robin" in a small temporary buffer (on disc with the System Tester or in SYMBOL core with the Maintenance Unit) for later playback only if something crashed. The buffer in the System Tester held

TABLE V—Examples of SUPERBUG Commands which Control the Maintenance Unit

```
; THIS IS A COMMENT CARD AND DOES NOT AFFECT THE MAINTENANCE UNIT IN ANY WAY
3; Pause. Press "START" on console to continue processing.
4; Enable printing of results on the line printer
5; Inhibit printing for greater test throughput
7; Skip the line printer to the top of a new page
8 00XXXX-DDDDDDDDDDDDDDDD-DDDDDDDDDDDDDDDD-etc.,   Load core with data
8 00XXXX-DDDDDDDDDDDDDDDD----------------------------------------etc.,   Load successive core locations with datum
8 -FFFFFFFFFFFFFFFF-1-4-30--AAAAAAAAAA00; Load beginning at address zero
9 00XXXX; Dump non-zero contents of memory on line printer
9;           Dump non-zero memory beginning at address zero
A 00XXXX-DDDDDDDDDDDDDDDD-DDDDDDDDDDDDDDDD-etc.,   Compare core with data
A -FFFFFFFFFFFFFFFF-1-4-30--AAAAAAAAAA00; Compare beginning at zero
B; Test core with all ones and all zeros, in that order
C XXXXXX-DDDDDDDDDDDDDDDD-P TN Ø; Memory cycle without check of results
C XXXXXX-DDDDDDDDDDDDDDDD-P TN Ø-RRRRRR-DDDDDDDDDDDDDDDD-RC TN; With check
D; System reset. Equivalent to pressing "RESET" on the console
E XXXXXX-DDDDDDDDDDDDDDDD-P TN; System Supervisor transmit without trace
E XXXXXX-DDDDDDDDDDDDDDDD-P TN 0 Q; Disc-synchronized System Supervisor transmit
E XXXXXX-DDDDDDDDDDDDDDDD-P TN 0 0 XMP CCCCCC; Trace after System Supervisor transmit
F M; Set new system test mode
```

the last 1600 operations whereas the Maintenance Unit's buffer held but the last 64; usually the most recent 10 to 20 were sufficient to analyze whatever problem caused the crash. As might be expected, this "instant reverse playback" feature gained heavy use since testing without the real-time printing of results speeded things up manyfold; only the last cycles which caused failure needed to be printed.

SUMMARY

The full-scale running SYMBOL system has demonstrated that:

1. A very high-level, general-purpose, procedural language and a multi-processing, multi-programming operating system can be implemented directly in hardware.
2. Design and construction techniques using only large two-layer printed circuit boards for all system interconnections, together with a functionally factored system result in an economical, serviceable, reliable, and "debuggable" system.
3. Effective testing techniques that connect an operating processor to an unproven system which has a bus-organized architecture can be developed to enhance and greatly simplify the debugging process

ACKNOWLEDGMENTS

The author wishes to acknowledge the help of the past and present members of the Digital Systems Research Department at Fairchild Camera and Instrument Corporation who contributed to the testing phase of SYMBOL. Steve Lundstrom designed the original System Tester and contributed to the development of the DEBUG language. Tom Cook assisted with the implementation of the Maintenance Unit. Eric Palmer wrote the SUPERBUG processor. The guiding hands of Rex Rice and William R. Smith were present throughout the entire project. And, of course, the hard work and enthusiasm of the other members of the department made the project succeed.

REFERENCES

1 R RICE W R SMITH
SYMBOL: A major departure from classic software-dominated von Neumann computing systems
AFIPS Conf Proc 38 pp 575-587 1971

2 W R SMITH R RICE G D CHESLEY T A LALIOTIS S F LUNDSTROM M A CALHOUN L D GEROULD T G COOK
SYMBOL: A large experimental system exploring major hardware replacement of software
AFIPS Conf Proc 38 pp 601-616 1971

3 G D CHESLEY W R SMITH
The hardware-implemented high-level machine language for SYMBOL
AFIPS Conf Proc 38 pp 563-573 1971

4 B E COWART R RICE S F LUNDSTROM
The physical attributes and testing aspects of the SYMBOL system
AFIPS Conf Proc 38 pp 589-600 1971

5 R B SEEDS W R SMITH R D NEVALA
Integrated complementary transistor nanosecond logic
IEEE Proc 52:12 pp 1584-1590 1964

6 R RICE
Functional design and interconnections: Key to inexpensive systems
Electronics 39:3 pp 109-116 1966

The Rice Research Computer—A tagged architecture*

by E. A. FEUSTEL

Rice University
Houston, Texas

INTRODUCTION

In this paper we report on a new computer with several novel features. These features are applications of the concept of tagged architecture, and although some of them are not unique to the Rice Research Computer (R-2), they focus our attention on this radical design form, its advantages and disadvantages. Since the work is still in progress, we limit this report to a discussion of the architecture and a few of its ramifications.

History of tagged architecture

The R-2 computer is an adaptation of a design of the Basic Language Machine (BLM)[1] of Iliffe. In his book and paper[2] he presents an argument for the utilization of a fraction of each memory word as tag bits. These tag bits are to be interpreted by the hardware as information about the data found in the referenced location, or its status with respect to the program or operating system.

The basic concept of tag bits is not new. Almost all computers employ a parity bit which the hardware uses to detect memory failure. In addition, many computers utilize a lock byte which limits access to an area of storage to the operating system or to those who have a key byte that opens the locked area.

Early machines also employed bits which were of special significance to the hardware. The Burroughs B5500 employed a flag bit to inform the hardware that the word at the location addressed possessed a non-numeric value which must be interpreted by the operating system.[3] The Rice Computer (R-1),[4] circa 1959, employed two bits for every word which could be set by the operating system or the programmer. These bits were used in an extensive debugging system wherein tracing, monitoring, or other procedures were carried out when a tagged data word or instruction was encountered.

Today the EAI8400 employs two tag bits for similar purposes. The Telefunken TR4 and TR440[5] employ two tag bits to denote the numeric type of data at an addressed location. The Burroughs B6700[6,7,8] and B7700 which were developed concurrently and independently of the R-2 employ three tag bits to identify types of numeric operands and special information used by the operating system.

What is new about Iliffe's concept is that it represents a rejection of the classical von Neumann machine in favor of something which may be better. In the von Neumann machine program and data are equivalent in the sense that the data which the program operates on may be the program itself. The loop which modifies its own addresses or changes its own instructions is an example of this. While this practice may be permissible in a minicomputer with a single user, it constitutes gross negligence in the case of a multi-user machine where sharing of code and/or data is to be encouraged.

Instead, Iliffe presents a different conjecture. All information which the algorithm needs to know about the data ought to be contained indivisibly in the data itself. For example, an algorithm to perform a *for*-loop on arrays ought to be the same whether the array is of length ten or length 100. Rather than record this information in a variable and use a loop with an index, Iliffe proposes to record the length of the array with the pointer to the array itself, as in Figure 1.

Rather than have several different algorithms for *add integer*, *add floating*, *add double precision*, and *add complex*, he proposes to make the data self-representing. An integer can only be used as an integer, a floating quantity as a floating quantity, etc. This idea is the fundamental difference between the class of machines represented by the BLM, the R-2, and the Burroughs B6700 on the one hand, and by those of more conventional architecture on the other.

Once the fundamental decision has been made to

* This work is supported in part by the Atomic Energy Commission under grants AT-(40-1)-2572 and AT-(40-1)-4061.

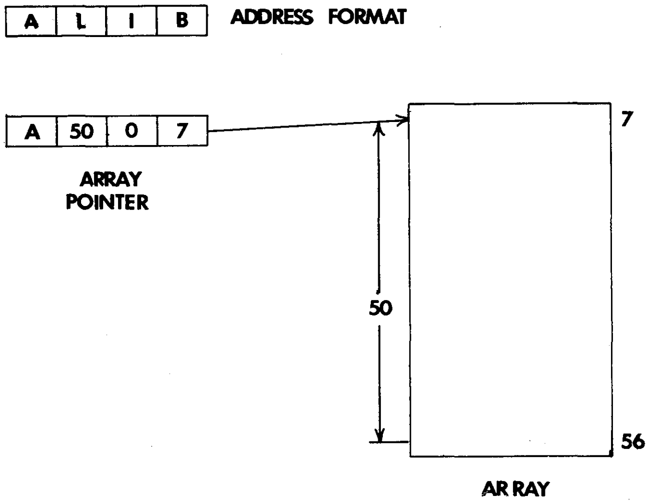


Figure 1—A typical array pointer and its array

adopt this goal in hardware, and the commitment to the use of memory bits for tags has been made, a great many benefits result. One of the most important is that the hardware, once informed of what each piece of data is, can perform run-time checks for consistency of data and algorithms, e g., bounds checking and type conversion.

General structure of the R-2

We now turn to the general structure of the R-2 as shown in Figure 2, which represents only a minor modification from the original design which began in January 1969. The system consists of three major asynchronous subsystems: a memory system of 24K 64-bit words of core memory, a Digital Equipment Corporation PDP-11 I-O controller, and the CPU complex. While the details of the first and second subsystem are of interest, we will not be concerned with them in this paper.

The CPU complex consists of a set of 64 16-bit scratch-pad memory circuits organized as 64-bit registers (designated XØ through X15), whose cycle time is approximately 40 nanoseconds, an arithmetic unit and a CPU. The latter two units are built from RCA ECL integrated circuits with typical delays of from 3 to 5 nanoseconds. This results in a typical add time for two 54-bit floating quantities on the order of 50-200 nanoseconds, and a multiply consisting of additions and shifts typically requiring 3 microseconds.

The address calculator in the CPU is one of the most important features of the system. It functions as an automatic base-bounds calculator and is responsible for the high security of the system programs. For every fetch from memory the address calculator is given four quantities, a base address B of 20 bits, an initial index I of 14 bits, a length L of 14 bits, and the element selector D of 14 bits. In 150 to 200 nanoseconds the calculator performs the following algorithm:

```
Temp:=D-I;
if Temp<0 then Low__error__1;
if Temp≥L then High__error__1;
Actual__address:=B+Temp;
```

I is a number between $-(2^{13}-1)$ and $(2^{13}-1)$ in one's complement form. D has a maximum value of $2^{14}-1$ and this is the largest segment which one can practically use. This should be sufficient for all but the largest one-dimensional arrays of data. The base address B is of sufficient size for any program (or memory) that we can currently foresee, on the order of 1 million words.

Data formats

Before we can discuss the operation of the CPU we must understand the various data formats which the R-2 can deal with. These formats are given in Figure 3. Each word currently consists of 62 bits of information. A two bit field in every word contains a parity bit Z and a write lock bit L. The remaining 60 bits may be divided into four classes of words: numeric words, control words, address words (or partition words), and instruction words. The first three classes contain six bits used for tags as Iliffe suggested. Four bits are used to distinguish types and two may be set by the programmer to generate interrupts. The type codes are listed in Table I.

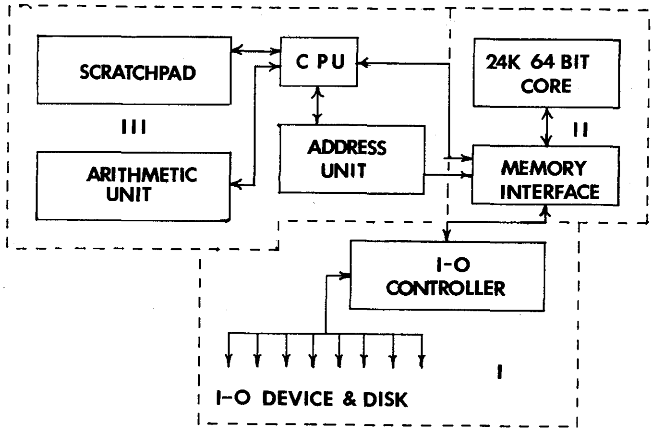


Figure 2—Subsystems of the R-2 Computer

The format of the various numeric types is of little interest. They are the same as those found in the original Rice Computer.[4] See Table I.

We should take greater note of the formats for address words. In addition to containing a base field (B), an initial index field (I) and a length field (L), each address word also contains an indirect reference field which indicates the type of the object in the block described by the address word. Two more bits are used. One indicates whether the block which is described is in core memory or in secondary storage (P). The other indicates whether the block may be relocated (Q).

Control words are an innovation. They are the only method of intersegment communication. They contain a 21-bit halfword pointer to an instruction (R-2 instructions are packed two per word). A mode field of 11 bits indicates the operating state of the computer at the time a jump to a subroutine is made. A two-bit condition code at the time the jump is made is stored in field C. Since the routine may be disk resident, a P field is provided to signal the routine's presence or absence in core memory. A chain field is provided in each control word. This field may be used to link together control words which are on the stack or are in a common linkage segment. Rather than scan storage linearly to find the previous control word, one merely follows the chain of links. Finally, a four-bit mark field may be used to indicate the level of the subroutine or the level of the subtask in the operating system. These marks are especially useful when employed with the control stack for exiting blocks and reestablishing an appropriate environment.

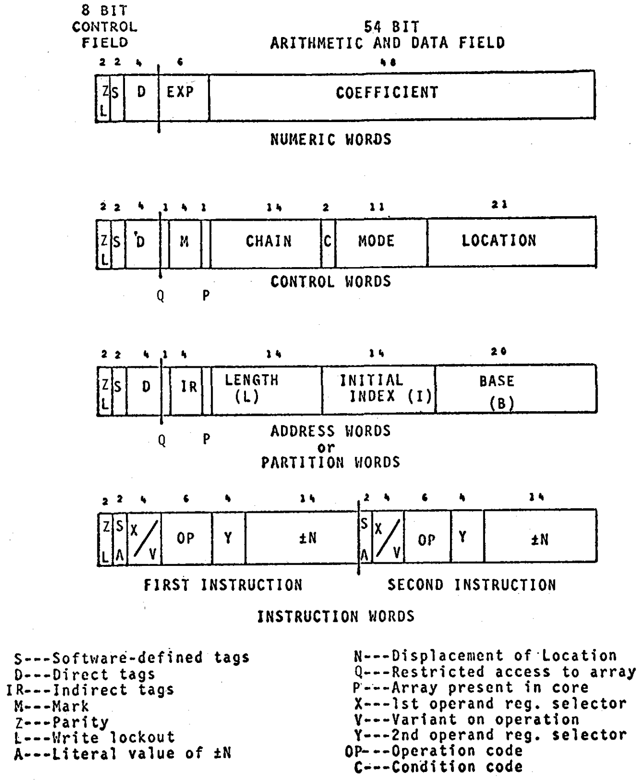


Figure 3—Diagram of R-2 word formats

TABLE I—Data Tag Assignment

TAG Number	*Meaning*
0000	MIXED OR UNTAGGED
0001	(unassigned)
0010	(unassigned)
0011	(unassigned)
0100	REAL, SINGLE PRECISION
0101	54 BIT BINARY STRING OR INTEGER
0110	DOUBLE PRECISION (real, fl. pt.)
0111	COMPLEX (two single precision fl. pt. words)
1000	UNDEFINED FOR NORMAL OPERATIONS
1001	PARTITION WORD
1010	RELATIVE CONTROL WORD
1011	ABSOLUTE CONTROL WORD
1100	RELATIVE ADDRESS, UNCHAINED
1101	ABSOLUTE ADDRESS, UNCHAINED
1110	RELATIVE ADDRESS, CHAINED
1111	ABSOLUTE ADDRESS, CHAINED

Processor facilities

Two major resources are available for use by the instruction unit. The first is the hardware stack and stacking mechanism. The second is the register set (previously described). The stack is maintained in memory and utilizes an address word held in register XØ as a stack pointer and bound. In order to utilize the address calculator in a consistent manner, the top of the stack is the location addressed by the base field of the address word and the bottom of the (accessible) stack is determined using the length field. Special words called partition words are stored by the operating system to denote the absolute beginning and end of the stack region or to point to a continuation of the stack in another segment (see Figure 4). Any word of the accessible stack within $2^{14}-1$ of the top may be accessed by an instruction. Partition words cannot be overwritten by normal stacking and unstacking operations.

Two hardware registers U and R constitute X1, the double length accumulator for arithmetic and logical operations. If X1 is loaded with an item of double precision or complex data, the second word is held in the R register. Special instructions are available to address the R register independently of U.

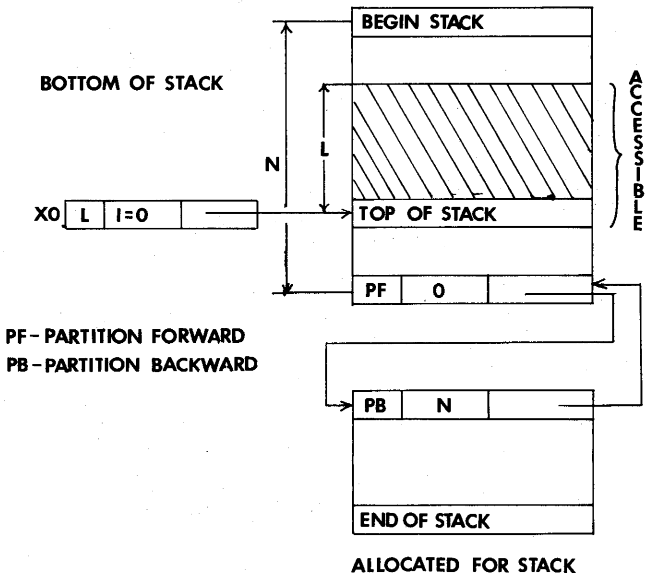


Figure 4—The stack pointer and stack layout

Registers X2 through X15 are implemented with the scratchpad memory integrated circuits mentioned previously, and a reserved, fixed core memory location is associated with each register to hold the second word of double word data when it appears. By recognizing the data tags "complex" and "double precision," the hardware automatically performs the appropriate double word transfers and storage so that no special programming is required. Thus, to the programmer X1 through X15 appear to be a set of general purpose double word registers. Since XØ is always used as the stack pointer, the hardware rejects (by interrupt) any attempt to store a word into XØ unless it is tagged as an address word.

All of the other registers except X15 may be used for temporary storage of numbers, addresses, or control words. X15 must contain an address word which describes where the interrupt vector is located. All interrupts transfer relative to this address. In the event of a catastrophe when there might not be an address word in X15, interrupts transfer to locations relative to absolute address 0.

Instruction formats

Every instruction contains a one bit software tag denoted by S and six bit function code field labeled OP in Figure 3. Each instruction also features an immediate bit labeled A, a numeric offset labeled N which may be plus or minus, and a register field Y. The remaining four bits labeled X/V in Figure 3 are interpreted differently in two classes of instructions. The first features a four bit result register field X. The second uses this field as a variant code for the operation, and for this class (which includes arithmetic and logic) the first operand is implicitly X1, which also contains the result.

Instructions are customarily written in the form S A X OP Y ± N or S A OP Y ± N where the function is to perform X OP $Y_{eff} \rightarrow X$. The first operand X is either stated explicitly or is implicit in the instruction. Y_{eff} depends on the contents of Y, the number N and the immediate bit A.

Rather than describe the operation of the addressing algorithm for Y_{eff} exhaustively by flow diagrams, we will describe instruction sequences for short algorithms. This will show the effect of Iliffe's conjecture, as well as illustrating the machine design.

EXAMPLE PROGRAMS

Example 1—Accessing vector elements

All elements must be accessed through an address word. Address words can be constructed by the operating system in a manner to be described in a following example. Suppose we have an address word in X2 which points to a vector in the manner of Figure 1, and suppose we wish to add the element with index ten of the array to a numeric quantity in X1. We could use the following instruction:

ADD X2.10 // Select the element of X2 indexed by 10 and add to X1. Suppose the initial index of X2, I(X2) is −4, the length 15, and the base address 1000. The sequence of computations would proceed as follows. The computer would check X2 and determine that it contained an address. It would issue B=1000, I=−4, L=15, and D=10 to the address calculator. The calculator would compute D-I (10−(−4)) or 14 which is greater than −1 and less than L. Since the element is within the vector, an address of 1014 would be generated and the element would be brought to the arithmetic unit. If the element is an integer it would be immediately added to the integer in X1 and the condition code would be set to reflect whether the result was less than, greater than, or equal to zero. If the element is real (fixed or floating point fraction), the computer would convert X1 to floating point form and perform the addition. If the element is anything but a numeric type, an exception occurs and an interrupt to a fixed location relative to the address in X15 takes place. If the instruction also invokes the auto-store option, denoted ADD→X2.10, the result would be stored back into 1014.

Example 2—Accessing array elements

On the R-2 we usually represent arrays in a tree structure. The first index is used to determine an element in a vector composed of address words. The second index is used with this address element to select an element from a second vector which is an address word and so on until the last index which is used to select the desired element. This kind of an array is illustrated in Figure 5. Because of the fact that each address word carries with it the length of the vector it addresses, such arrays may be uniform or nonuniform as desired. They may also be so large that only one vector of data will fit in core memory at any time.

Two different methods of addressing such arrays can be used. These methods are considerably more efficient than that used on the Burroughs 6700 because of the scratchpad registers X2-X14 which are available. One method involves element selection as in the first example. It is generally used when we wish to select only $X_{i,j,k}$th element of an array rather than to deal with every element. The following sequence of instructions indicates how this may be accomplished.

Suppose X2 contains the address word pointing to the vector and we desire to select $Z_{3,1,5}$ and assume that the initial indices are 0. Then we could use:

X2 DOT = 3 // Replaces the contents of X2 with the third element of first level tree.
X2 DOT = 1 // Replaces of the contents of X2 with the first element of second level.
X2 MOD = 5 // Generates the address of $Z_{3,1,5}$ in X2.
ADD → X2 // Adds (X1) to $Z_{3,1,5}$ and autostores back into the array.

This set uses the immediate address form of the instruction.

An alternative form might employ a vector of subscripts. Suppose that X3 contains the address word of a vector of subscripts to be applied to Z, i.e. (3, 1, 5). The following sequence indicates how this may be done.

X2 DOT X3.1 // Obtains the third element of first level.
X2 DOT X3.2 // Obtains the first element of second level.
X2 MOD X3.3 // Generates a pointer to $Z_{3,1,5}$.
ADD →X2 // Adds (X1) to $Z_{3,1,5}$. and autostores.

This sequence of computations is as follows. The first element of X3 (since I(X3) = 0) is selected and brought to the CPU. It has the value 3. If X3 had contained a number originally, 1 would have been added to that number and the resultant would have been used. This value is then used to obtain the third element of Z's first level subtree (denoted $Z_{3,*,*}$). This is left in X2. The next operation obtains an address word which is the first element of Z's second level tree of the third branch ($Z_{3,1,*}$). The next operation indexes this address to make its location field point to the desired element. This element is presumably a number (integer, real, complex, or double precision). It is added to the contents of the U register (X1), and the result placed in X1 and in $Z_{3,1,5}$, thus smoothly implementing the ALGOL 68 statement: Z[3, 1,5]+: = V; where V was the contents of X1. If the value of (X3.1), (X3.2), or (X3.2) had not been a number then an exception would have occurred.

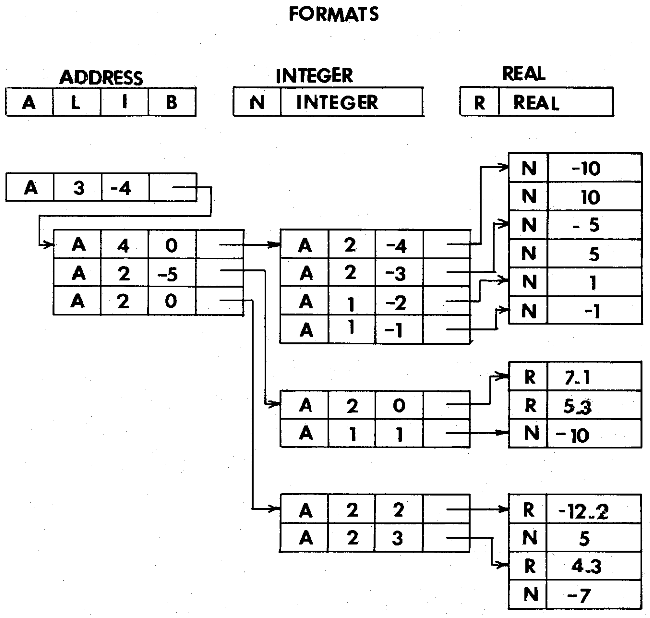


Figure 5—A nonuniform three dimensional array

Example 3—Array processing

In some cases vector processing is desired. For this purpose a different kind of access is desired. In this mode one systematically examines all the elements of an array or vector in turn, and performs some operation on them. For example, to sum a vector pointed to by an address word in register X2, one might use the following sequence of instructions. Assume (I(X2)) equals 0.

```
         X3 LOAD    4    // To sum five elements of
                            vector X2
         X1 LOAD    0    // Initialize X1 to Zero
AGAIN       ADD X2.0     // Add first element of
                            vector
         X2 MOD   =1     // Adjust X2 to point to
                            the next element
         X3 JGE AGAIN    // Continue to CONT if
                            X3 is zero or negative
CONT                     // Decrement and jump to
                            AGAIN otherwise.
```

This sequence loads X3 with the number 4 and X1 with the number 0. The next instruction causes the number pointed to by the B field of X2 to be added to the number in X1. If the length field is set to −1, an exception will occur. The next instruction uses the literal 1 to decrement the L field of the address in X2 and simultaneously increase the B field by 1. If L is less than zero an exception will occur. The next instruction examines X3. If it is a number less than or equal to 0, the next sequential instruction is taken. Otherwise the number is decremented by one and control passes to AGAIN. Using this sequence the first five elements of the vector pointed to by X2 are summed. If there are fewer than six elements in this vector, an exception occurs. If there are exactly five elements in the vector when the program gets to CONT, L(X2) equals zero. An attempt to do X2 MOD=1 again will cause an error exception. Otherwise the new L(X2) is five less than before and the new B(X2) is five more than previously.

A shorter sequence may be used if it is desired to sum all the elements of the vector. Here the programmer need not even know how long the vector is.

```
         X1 LOAD  =0     // Set X1 to zero
AGAIN       ADD  X2.0    // Add elements to X1
         X2 JNL  AGAIN   // See the discussion below
```

The last instruction checks to see if L(X2) is greater than 0. If it is it performs X2 MOD=1 and transfers control to AGAIN. If it is not, control passess to the next instruction.

As a final example of the power of this approach, assume that we have three arrays A, B^T, and C and that we desire to compute

$$C_{i,k}=\sum_{j=0}^{n} A_{i,j}B_{k,j}{}^{T}.$$

This can be calculated simply in the following routine assuming X2 is an address word pointing to A, X4 is an address word pointng to B^T, and X6 is an address word pointing to C.

```
INIT     X9 COPY   X4      // Copies (X4) to X9
BEGIN    X7 LOAD   X6      // Get ith subtree of
                              C
FIRST    X3 LOAD   X2      // Get ith subtree of
                              A
         X5 LOAD   X4      // Get kth subtree of
                              $B^T$
            ZERO   X7.0    // Put zero in $C_{i,k}$
SECOND   X1 LOAD   X3.0    // Get $A_{i,j}$
            MUL    X5.0    // Multiply by $B_{k,j}{}^T$
            ADD→   X7.0    // Add and autostore
                              to $C_{i,k}$
         X5 MOD    =1      // Next consider
                              $B_{k,j+1}{}^T$
         X3 JNL SECOND     // Next consider
                              $A_{i,j+1}$
CONT                       // if no more j con-
                              tinue here
         X4 MOD    =1      // Next consider
                              $B_{k+1,j}{}^T$
         X7 JNL FIRST      // Consider $C_{i,k+1}$ if
                              any left
         X6 MOD    =1      // Consider $C_{i+1,k}$
         X4 COPY   X9      // Start over with
                              $B_{0,0}$
         X2 JNL BEGIN      // Consider $A_{i+1,j}$ if
                              any left.
```

This routine destroys pointers located in X2, X4, and X6. The steps

```
            MUL    X5.0
         X5 MOD=1
```

may be combined into MUL! X5 which uses a variant option for the arithmetic operation code to modify the address word in X5 after the element has been fetched.

Example 5—Use of the stack

The stack may be used for intermediate storage in the following manner. It is first necessary to get the operand in a scratchpad register. Suppose we wish to stack X5. Then we write X5 STORE XØ. The contents of X5 is pushed onto the stack. On the other hand, X5 STORE XØ.7 stores the contents of X5 in the seventh location of the stack without altering other elements. If there are less than seven elements on the accessible stack an exception occurs. The top element of the stack may be changed without pushing by the use of X5 STORE XØ.1.

Elements are removed from the stack in an analogous manner. For example, ADD XØ adds the top element of the stack to the accumulator and pops the top element. ADD XØ.1 adds the top element of the stack but does not pop it; ADD XØ.7 adds the seventh element of the stack without affecting the stack.

This stack arrangement affords many conveniences in arithmetic operations. An example of this is the instruction save and fetch. Take as an example X2 SVF X3.12. This instruction gets the twelfth element of the array pointed to by X3, after saving the old value of X2 on the stack, and places the new value in X2. In compiling, the instruction X1 SVF Xi.N will occur frequently. Intermediate results can be saved on the stack for later use. Alternatively, when the value is to be used many times they can be stored in a register using COPY.

The stack is also useful in control actions. The instruction JUMP AND SET MARK, e.g., 4 JSM LABEL, causes a control word to be made up pointing to the next sequential instruction. This control word contains the current mode, the current condition code, and the mark specified by the JSM instruction, in this case 4. The chain field is loaded with the current value of L(XØ). The resulting control word is pushed on the stack. XØ is then updated to reflect a new stack regime. L(XØ) is set to 0 and B is set to the location that is one less than that occupied by the control word. This stack regime is completely disconnected from the prior one; there is no way save through the registers or memory constants to reach any of the members of the previous stack regime (see Figure 6). One can return to the previous environment by the use of the return instruction: RET 4. The previous control word is found by examining the address B(XØ)+L(XØ)+1. This address contains the last control word or a partition word pointing to another section of the stack regime or a partition word marking the absolute beginning of the stack. If a control word occupies this position, B(XØ) is set to point to this address. The chain field of the control word is copied into L(XØ), the mode field to the mode register, the condition code to the condition code register, and the 21 bit address to the program counter. If the mark field is less than the literal used with the return instruction, the computer resumes processing. Otherwise the process described above is repeated until a partition word marking the absolute beginning of stack is encountered. If the beginning of stack is encountered, an error exception will result. This form of return has the advantage that it is efficient and can be used to return several levels in a block structured language.

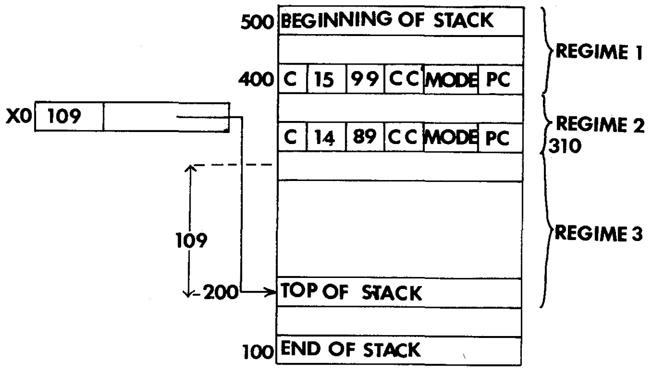


Figure 6—The use of the stack in programming systems

Relative addressing

In an earlier section we stated that all addressing was done via an address word. In fact this is not quite true. Addresses may be relative to the program counter or to the location in which the address word or control word is stored. One may then jump minus five halfword instructions. This feature was available on the R-1 in 1959 and made possible the development of very efficient relocatable code. It is frequently found on machines today. One can also access fullword data in the same manner so that constants can be stored with the code.

In the same way relative codewords and addresswords can be used to address other quantities. The address of the location in which the address or control word is stored must be determinable by the computer. An offset relative to this address is used to point to the jump location or to the data. This feature makes possible relocation of large blocks of data in a very efficient manner and minimizes the number of address words which contain absolute addresses. This is important because it drastically reduces overhead in the reorganization of storage. Only a few locations need to be modified to relocate all programs and their data.

Chained Addressing

A second kind of addressing is also provided. Chained addressing provides for efficient parameter passing mechanisms, particularly of the call by reference variety. The addressing algorithm is modified to include indirection. When a chained address or control word is encountered the machine causes an indirect reference through the address or control word to the next quantity. A counter is employed to assure that chaining beyond 32 levels is not allowed. Special instructions are employed to defeat chaining for loading and storing. These instructions allow complete control of the addressing mechanism.

Miscellaneous instructions

The R-2 is designed to be used with compilers rather than assemblers. It has many useful miscellaneous instructions including reverse divide, variants of pre- and post-complemented addition and logical operations, and integer and floating multiplication and division. Various instructions allow extraction and replacement of the predefined data and instruction fields. A large number of shift, bit count, and test instructions complement the arithmetic and logical set of instructions, thus facilitating the development of operating systems and compilers.

Ramifications of tagged architecture in the R-2

The use of tagged architecture has many ramifications. In a future paper we will discuss them more fully. Here we will comment on a few of interest to those who write or use compilers and operating systems or who must debug programs.

Compilers can be made simple and more efficient. Since tags indicate what each numeric quantity is and since the hardware will correctly perform the appropriate operation on all legal combinations of data, the compiler need not deal with semantic operations referring to basic types of identifiers. This is now handled conveniently at run-time. The problem of temporary storage is largely solved by the stack whose implementation is greatly facilitated by the address calculator and the tagged address type. The tagged registers allow simple manipulation and mechanization of vector and array operations and allow dynamic variables much more freedom than do previous machines. They also permit the optimum calculation of expressions of type *address* and type *numeric.* Finally, tagged addresses and numbers greatly simplify the problems of run-time systems for use with particular compilers; note that even undefined quantities have a distinct representation.

Operating system design is facilitated by tagging. The difference between a label and an address can be determined at run-time. This means that one cannot jump through an address word or a number but only to approved points in a subroutine via a control word. Attempting to do otherwise produces an interrupt.

Secondly, addresses can only be manipulated by means of a special set of instructions. The more powerful instructions may be denied to the user and he may be given the use of MOD, TAG, and LIM. TAG is an instruction with an immediate operand. The compiler monitors all TAG instructions assuring that no unauthorized user can construct illegal address or control words. Any user may employ MOD or LIM. They modify an address to point to a subset of the elements in an addressed space. Since an unprivileged user may never generate an address outside the initial space to which he has been given access, protection of user programs is enhanced if not insured. This protection mechanism appears to be exactly what is required for recursively defined operating systems.

The design and use of debugging systems is greatly simplified. A program can be written to dump core memory using the type of each datum. This means that complex, double precision, floating, integer, and undefined types can be used to interpret the data contained in the cells. This can be used in the analysis of dumps. Addresses and labels are also distinct in this scheme; the fact that they are not in other systems has been a recurring problem for those who must debug.

Dynamic debugging is also easily implemented. By the use of symbolic locations, relative locations, or absolute locations, data or instructions may be tagged with software tags. Whenever such tagged data is encountered, an interrupt occurs. The programmer may supply his own programs to analyze or monitor the data values which are encountered. The software tag in each of the instructions may also be set to cause interrupts related to tracing any or all control actions. Again the user may write a program to analyze the results. Finally since the tag bits may be set either at compile time by the compiler, or arbitrarily at run-time by the user through the operating system, code which is verified as being correct need not be recompiled, thus simplifying and expediting the process of debugging.

SUMMARY

This paper has reported the state of the Rice Research Computer in its development. It has emphasized the features of the R-2 which arise from accepting the principle of tagged architecture. For a particular implementation, we have shown by example the power of tagged architecture in application to compilers, operating systems, and debugging systems.

ACKNOWLEDGMENTS

I would like to acknowledge the work of the designers of R-2 including John Iliffe, I.C.L. (London, England), Walter Orvedahl (Colorado State University), Dr. Sigsby Rusk (Rice University), Douglass DeLong (Graduate Student, Rice University), Dr. Ernest Sibert (Syracuse University), and Dr. Robert C. Minnick (Rice University). Also, I owe thanks to Dr. J. Robert Jump and the graduate and undergraduate students at the Rice University Laboratory for Computer Science and Engineering who have made and are making the machine a reality.

REFERENCES

1 J K ILIFFE
Basic machine principles
American Elsevier New York 1968

2 J K ILIFFE
Elements of BLM
Computer Journal 12 August 1969 pp 251-258

3 *A narrative description of the Burroughs B5500 disk file master control program*
Burroughs Corporation Detroit Michigan Revised October 1966

4 *Basic machine operation*
Rice University Computer Project Houston Texas January 1962

5 *TR440 eigenschaften des RD441*
AEG Telefunken Manual DBS 180 0470 Konstanz Germany March 1970 (German)

6 *Burroughs B6500 information processing systems reference manual*
Burroughs Corporation Detroit Michigan 1969

7 J G CLEARY
Process handling on the Burroughs B6500
Proceedings of the Fourth Australian Computer Conference Griffin Press Adelaide South Australia 1969 pp 231-239

8 E I ORGANICK J G CLEARY
A data structure model of the B6700 computer system
SIGPLAN Symposium on Data Structures and Program Languages ACM New York New York 1971 pp 83-145

A generative CAI tutor for computer science concepts*

by ELLIOT B. KOFFMAN

University of Connecticut
Storrs, Connecticut

INTRODUCTION

Limited progress has been made in software for computer-assisted instruction. Frame-oriented CAI systems have dominated the field. These systems function as mechanized programmed texts and utilize the computational power of the computer to a minimal extent. In addition, they are difficult to modify and tend to provide a fairly fixed instructional sequence.

The conventional frame-oriented CAI system is organized as a series of frames. A frame may present a piece of information and/or ask a question. The questions are normally objective and often are of the multiple-choice type. The frames are usually linked in a sequential fashion and a student will cycle through them one at a time. Frames may be presented on a teletype, a graphical display, a slide projector, via an audio track, or any combination of the above.

There are severe problems inherent in systems of this type. All questions must be specified by the course-author as well as a set of anticipated student responses to each question. If branching is to occur, explicit instructions must be given indicating the performance criteria for a branch and the new continuation point in the program.

Since everything must be specified in advance, extensive time must be spent in preparing course material for presentation. Furthermore, once programmed this material has very little flexibility. Modifying the set of questions to be asked of the student or the material to be presented is a major undertaking and much reprogramming must be done.

This type of system is not very useful in teaching quantitative courses. Subject areas such as engineering or the physical sciences are concerned with teaching techniques of problem solving. Problem solving competence is often acquired through a process of "learning by doing." Consequently, it is essential that the CAI system be capable of presenting a wide variety of problems and solutions to the student. Reprogramming each problem and its solution in a manner suitable for presentation by CAI would be extremely inefficient.

It is precisely for these reasons that generative CAI systems have recently become of great interest. Generative systems are capable of generating a series of questions (and answers to these questions) as a function of the student interaction. These systems can be divided into two classes. Those which are oriented toward the "soft-sciences" and textual material and those which are more concerned with numerical manipulations and quantitative material.

Carbonell[1] and Wexler[2] have designed generative CAI systems which have been used to teach concepts in geography. These systems are organized around an information structure or network. Carbonell uses the semantic network developed by Quillian.[3]

Once the information network has been specified, these systems are capable of generating a sequence of questions for a student. As each question is generated, the answer is retrieved for comparison with the student's response. If the student is incorrect, Wexler's system is capable of providing individualized remedial comments. This would consist of either a correct and relevant statement using the student's incorrect answer or a systematic presentation of the steps performed by the system in deriving the correct solution. Both these systems allow the student to interrupt and pursue topics which interest him at greater depth.

The potential for incorporating generative CAI in the "hard sciences" is extensive. Algorithms for solution of classes of problems could be incorporated into CAI systems. In some cases, solution techniques might be sufficiently complex that heuristic programs would be necessary. Examples of the latter case would be teaching symbolic integration or proving theorems. In

* This research is sponsored by the Connecticut Research Commission under Grant RSA-71-7.

any event, CAI systems organized around a set of algorithms would have the capability to generate and solve a wide range of problems.

An extensive project in the subject area of analytical geometry has been described by Uttal.[4] His system is capable of generating twelve problem types which are representative of the problems found in an analytical geometry course. These problems usually involve an expression or graphical representation of a particular conic section. The expression is obtained from the general quadratic equation: $AX^2+BY^2+CX+DY+E=0$.

The required expression is obtained by setting certain coefficients to 0 and selecting the others at random. The complexity of the equation generated depends on the constraints imposed on the coefficients. For example, to generate circles centered at the origin, $A=B$ and $C=D=0$.

Associated with each of the twelve problem types is an answer routine. The routine which determines if a randomly generated point (x,y) falls on the locus represented by a randomly generated equation simply plugs this point into the equation. The expression generator itself is used as the answer routine when the student is asked to supply the equation for a conic section with given standard characteristics.

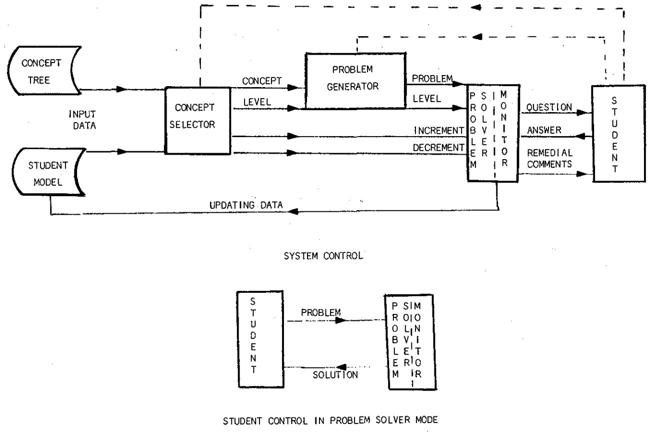
The following sections will describe a generative tutor that has been used in an introductory computer science course. It has been used to teach concepts of digital circuit design as well as to introduce students to machine language programming. Because of the large number of concepts covered, an intelligent "concept selector" has been designed which attempts to tailor the current instruction each student receives to fit his past performance record.

GENERATIVE CAI IN DIGITAL SYSTEMS

The system is designed to be extremely flexible in that it can completely control the progress of a student through the course, selecting concepts for study on an individual basis and generating problems. Alternatively, the student can assume the initiative and determine his own areas of study and/or supply his own problems.

In addition, the system also operates in a "problem-solver" mode. In this mode, the student specifies the concept area and his problem, and the system will crank out the solution without further interaction. It is anticipated that students in later courses and the digital laboratory will utilize this mode for solving complex minimization problems and determining the relative merits of different state assignments.

Figure 1 is a block diagram of the system functions. Subsequent sections of this paper will describe how these functions are accomplished. As has been mentioned, the student can assume the initiative and bypass the Concept Selector and/or Problem Generator (indicated by dashed lines from Student box). The bottom part of Figure 1 shows the student exercising full control in the problem-solver mode.



SYSTEM CONTROL

STUDENT CONTROL IN PROBLEM SOLVER MODE

Figure 1—System block diagram

When the system is in control of the interaction, it attempts to individualize the depth and pace of instruction presented to each student. A model of each student is kept which summarizes his past performance in each of the course concepts. Table I shows the contents of a student record.

In addition, the system is supplied with a concept tree which indicates the degree of complexity (plateau) of a concept and its relationship with other concepts in the course. A sample tree for the concepts currently covered is shown in Figure 2. This, of course, represents the author's interpretation of the relationship between course concepts. There are alternate interpretations which are just as valid.

The system uses each student's record to determine how quickly he should progress through the tree of concepts, the particular path which should be followed, the degree of difficulty of the problem to be generated, and the depth of monitoring and explanation of the problem solution.

Figure 3 is a flow-chart of the overall operation of the system.

PROBLEM-SOLVER/MONITOR

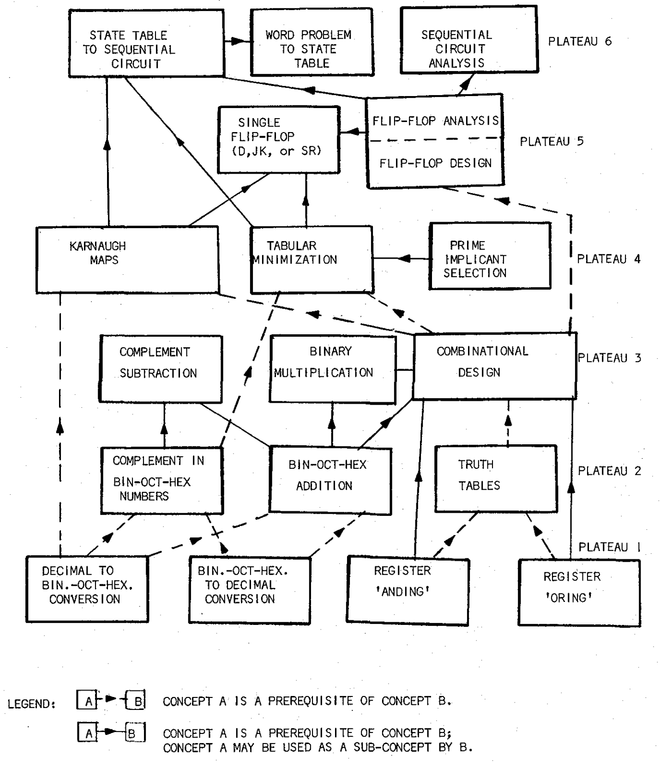
The course is organized as a set of solution algorithms. Normally there is a single algorithm for each major concept of the course. The algorithms solve the problems much as a student would, breaking each problem

TABLE I—Student Record

Student Name E B Koffman Master Ave. 1.8 Current Plateau 5

	Concept #1	Concept #2	. . .	Concept #30
Level	2.2	1.5		0.5
Last level change	.1	.5		−.2
Weighted Av. level change	.5	.6		−.1
Date of last call	3/15	3/17		4/20
Sequential order of last call	25	31		48
No. of times called in 0-1 range	2	2		3
No. of times called in 1-2 range	2	1		0
No. of times called in 2-3 range	0	0		0
No. of problems generated	2	3	. . .	3

down into a series of sub-tasks. After each sub-task is accomplished, a decision is made whether or not to question the student on this part of the problem solution. This decision is based on the student's current level of achievement (a number between 0 and 3) in the concept.


STATE TABLE TO SEQUENTIAL CIRCUIT
WORD PROBLEM TO STATE TABLE
SEQUENTIAL CIRCUIT ANALYSIS
PLATEAU 6
SINGLE FLIP-FLOP (D,JK, or SR)
FLIP-FLOP ANALYSIS
FLIP-FLOP DESIGN
PLATEAU 5
KARNAUGH MAPS
TABULAR MINIMIZATION
PRIME IMPLICANT SELECTION
PLATEAU 4
COMPLEMENT SUBTRACTION
BINARY MULTIPLICATION
COMBINATIONAL DESIGN
PLATEAU 3
COMPLEMENT IN BIN-OCT-HEX NUMBERS
BIN-OCT-HEX ADDITION
TRUTH TABLES
PLATEAU 2
PLATEAU 1
DECIMAL TO BIN.-OCT-HEX. CONVERSION
BIN.-OCT-HEX. TO DECIMAL CONVERSION
REGISTER 'ANDING'
REGISTER 'ORING'
LEGEND:
A B CONCEPT A IS A PREREQUISITE OF CONCEPT B.
A B CONCEPT A IS A PREREQUISITE OF CONCEPT B; CONCEPT A MAY BE USED AS A SUB-CONCEPT BY B.

Note: The relation "is a prerequisite of" is transitive (A is a prerequisite of B, B is a prerequisite of C, implies A is a prerequisite of C)

Figure 2—Concept tree

If the student is questioned, then his answer is compared with the system's solution. If the student is correct, his level is increased; if he is incorrect, he will receive a remedial comment explaining the correct solution procedure and his level for that concept will be decreased. The higher a student's level, the fewer questions he will be asked. When the student reaches a level of 3 in a concept, the system will solve subsequent problems dealing with this concept for him.

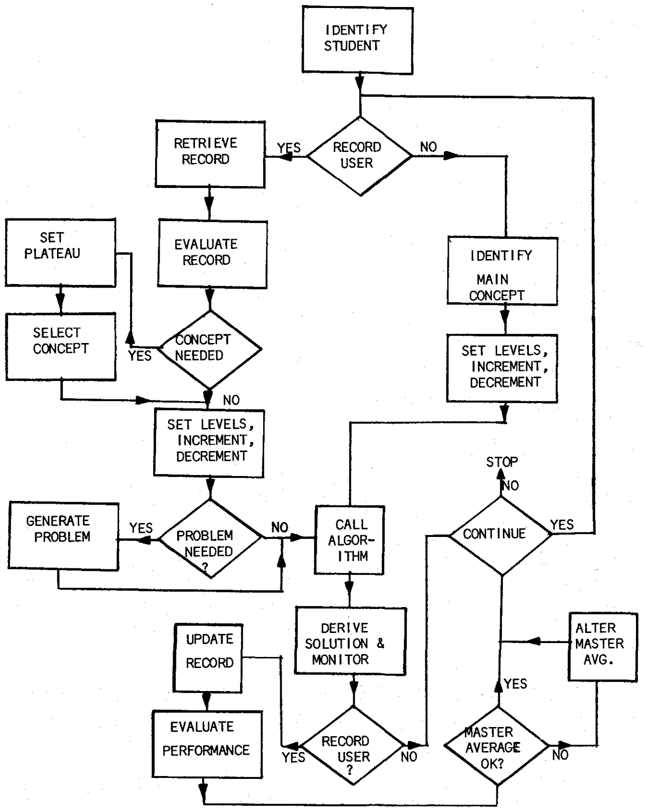


Figure 3—System flow chart

Table II presents examples of different degrees of interaction with a student in an octal addition problem. The first character of all student inputs is underscored. A single algorithm solves binary, octal, and hexadecimal addition problems. The decimal sum of each pair of digits and old carry is first calculated. The corresponding new carry and sum digits in the problem base is then determined.

A student with level less than 1 is quizzed on all details of this algorithm; while, the initial decimal calculation is skipped for students with level between 1 and 2. The student with level between 2 and 3 is asked only for his final result; whereas, the final result is generated for a student with level equal to 3. The

TABLE II—Different Degrees of Interaction

Problem: Form the sum of 75.3, 2.7 in the base 8.

LEVEL RANGE 0-1
Modify 2.7 for addition to 75.3
02.7
Correct. The rightmost digit is digit #1.
What is the decimal sum of digit #1
of 75.3 and digit #1 of 02.7?
8
No. Their decimal sum is 3+7 or 10.
What is digit #1 of the sum in the base 8?
1
No. The decimal number 10 is equivalent to 12 in the base 8. Digit #1 is 2.
What is the carry to digit #2?
1
Correct. What is the decimal sum of digit #2 of 75.3 and digit #2 of 02.7 and the carry from digit #1?
⋮
LEVEL RANGE 1-2
The rightmost digit is digit #1.
What is digit #1 of the sum in the base 8?
2
Correct. What is the carry to digit #2?
0
No. 3+7=12 in the base 8.
The carry to digit #2 is 1.
What is digit #2 of the sum in the base 8?
⋮
LEVEL RANGE 2-3
What is the complete sum of 75.3, 02.7 in the base 8?
77.2
No. The sum of 75.3, 02.7 is 100.2 in the base 8.
LEVEL RANGE≥3
The sum of 75.3, 02.7 is 100.2 in the base 8.

amount of interaction a student receives will vary as the problem solution progresses and his level changes from one range to the next.

In this manner, the amount of monitoring is dynamically adapted to fit a student's current level of achievement. Often a solution algorithm will utilize an algorithm dealing with another course concept as a subroutine. When this happens, the student's level in the sub-concept will control the depth of instruction. If he has mastered the sub-concept (level ≥ 3), this portion of the solution will be provided for him.

When a student response is incorrect, he receives an individualized remedial comment which explains the system's derived answer. For example, in a problem utilizing the Flip-Flop Design algorithm, the student would be given the current state and desired next state and asked to supply the required values of the Flip-Flop excitations. Table III presents some of the alternative forms of feedback which might be presented depending on a student's level and his answer.

The correct solution in this case is derived through a table lookup. System response *A* is automatically generated if one of the excitations match and the correct answer for the non-matching excitation is "*D*" (a don't care). System response *B* is presented for all other types of errors. The additional information presented to students with level <1 is obtained by calling the Flip-Flop analysis algorithm in the problem-solver mode. This algorithm determines the next state which would result from the current state if the student's values of the Flip-Flop excitations were applied.

The magnitudes of the increment and decrement for correct and incorrect answers respectively are also individualized to fit the student's past performance. They are calculated for the particular concept which has been selected as follows:

$$\text{Increment} = K2 \cdot K1$$

$$\text{Decrement} = K2/K1$$

where

$$K1^2 = 1 + W_N + \Delta L \text{ and } .5 \leq K1^2 \leq 2, \; K2 = 1 - .5R$$

$\Delta L =$ last change in level

$W_N =$ weighted average level change for N uses of a concept

$$W_N = \frac{(N-1) \cdot W_{N-1} + 2 \cdot \Delta L}{N+1}$$

$$R = (\text{Number of uses of concept at present level range})/N \qquad (1)$$

If the student's level in a concept has been consistently increasing, W_N and ΔL will be positive and $K1^2$ will tend to be large. $K1^2$ represents the ratio of Increment to Decrement. Consequently, a student who has been performing well in a concept will be rewarded more for a correct answer than he is penalized for an incorrect one. Thus, he is receiving the benefit of the doubt for an occasional incorrect answer and progresses more rapidly to the problem-solver mode for that concept. The opposite is true for a student who has been performing poorly. This will enable him to drop quickly to a lower level at which he will receive further monitoring and instruction if he continues to answer incorrectly.

R represents the stability of the current level for the concept in question. *K*2 assumes values between .5 and 1.0, the higher values representing a greater degree of instability. Consequently, when the current level is in a relatively unused or unstable level range, the Increment and Decrement will both tend to be larger than when the current level appears to be a stable one. This makes it easier for a student to move out of a new level range; whereas, he must answer more questions correctly (or incorrectly) if he is to move out of a level

TABLE III

QUESTION

The current state of JK Flip-Flop 1 is 0.
The next state should be 1.
What are the values of $J1$, $K1$?

STUDENT ANSWER: $J1=1$, $K1=1$

SYSTEM RESPONSE A: Your answer is correct, however, a better answer would be $J1=1$, $K1=D$.
(D symbolizes a "don't care" condition)

The remedial feedback generated for a wrong answer to the above question follows:

STUDENT ANSWER: $J1=D$, $K1=1$

SYSTEM RESPONSE B: Wrong. To bring a JK Flip-Flop from the 0 to 1 state, set $J=1$ and $K=D$.

ADDITIONAL RESPONSE FOR LEVEL<1: Your Flip-Flop excitations would cause the next state to be indeterminate.

range which has become established as typical of him for a particular concept.

The values of Increment and Decrement calculated in this manner will have a minimum of .35 and a maximum of 1.4. These are multiplied by a fraction associated with each question. A typical value of this fraction is .2 which means the magnitude of the change in level per question is normally between .07 and .28.

The entry level of a student during his initial use of a concept is set equal to $.5+2-M$, where M is his master average. The significance of a student's master average will be discussed in the next section.

CONCEPT SELECTOR

Since there are a large number of concepts available for study, the system attempts to select the next concept in such a way as to make optimal use of the student's time. The goal is to pace the student through the concepts quickly enough so that he does not become bored or unmotivated and yet not so fast that he becomes unduly confused.

There is no set order in which the concepts are selected nor is there a set level of achievement which every student must exceed in order to advance. The algorithm attempts to individualize concept selection through examination of the student's performance record.

Each student is assigned a master average when he first logs onto the CAI system. This could be a function of his I. Q. or class standing. (In the past, each student has been arbitrarily assigned an initial master average of 2.) This value changes as the system gains experience with a student.

A student's master average controls the speed with which he jumps from one plateau of the concept tree to the next. In order to jump to the next higher plateau, the average of his levels of achievement in all concepts at and below the current plateau must exceed his master average. Consequently, the lower a student's master average, the faster he will progress.

Each student's master average is updated after the completion of a problem as follows:

$$\Delta M = -K(\Delta L + W_N)(1.1-R) \qquad (2)$$

where ΔL, W_N, and R have been defined previously. (See Equation 1.)

$K=1$ if the concept is from the student's current plateau

$K=2$ if the concept resides at a higher plateau and $\Delta L>0$.

$K=2$ if the concept resides at a lower plateau and $\Delta L<0$.

$K=0$ for all other cases.

Since R cannot be >1, ΔM and $(\Delta L+W_N)$ will be opposite in sign.

If the system selects the concept from the student's current plateau, K will be 1. If the student's level increases and W_N is also positive, his master average will decrease. If the student has selected the concept from a higher plateau and $\Delta L>0$, the magnitude of the decrease is doubled as this indicates the student is ready to progress more quickly. If the concept is above his current plateau and he does poorly ($\Delta L<0$), he is not penalized by an increase in his master average ($K=0$). However, if the concept is a remedial one (below his current plateau) and $(\Delta L+W_N)$ is negative, the increase in his master average is twice what it would be for the same performance in a concept from his current plateau.

The effect of the term $(1.1-R)$ is to cause those changes, which occur when in a relatively new level range, to have a greater influence on the master average. This is reasonable since a student who performs

TABLE IV—Sample Problems Generated

1. Convert the decimal number 65.75 to the base 8.
2. Calculate 75.3−24.56 in the base 8 using 8's complement subtraction.
3. Derive the truth table for $((P \vee (\neg Q)) \uparrow (\neg (R \wedge Q)))$
4. Design a combinational circuit such that:
   Input word A is 2 bits long. Input word B is 2 bits long.
   The output X=A+B. The output Y=1 if A<B.
5. Minimize a function which has 0, 1, 3, 4, 5, 10, 11 as minterms and 2, 8, 14 as don't cares using the Karnaugh Map Method.
6. Minimize a function which has 2, 4, 5, 10, 11 as minterms using the tabular method.
7. Find the excitation equations for a JK Flip-Flop such that:
   Input word A has maximum value 7.
   The Flip Flop is in state 1 if A=0, 2, 4, 6.

well after reaching a new level range is indicating that he is not disturbed by the decrease in interaction and is ready to advance at a faster pace. While a student who does poorly may be in over his head and might prefer it if the pace were slowed down. In any event, the magnitude of the change in master average may not exceed .2.

After the master average is updated, the student's plateau is determined by comparing his new master average with the average level of all concepts at and below his current plateau. If this average level exceeds his master average his plateau is increased by one and the comparison is repeated.

Once the student's plateau has been determined, the system selects a set of candidate concepts from this plateau and those below it if necessary. In order to qualify as a candidate concept, the average of the student's levels of achievement in all prerequisites for this concept (as determined from the concept tree) must exceed his master average. If this is not the case, the prerequisite concept in which the student has the lowest level is selected as a candidate in its place. This provides for automatic review of selected concepts at lower plateaus.

The system then chooses one concept from among the candidates. Each concept is evaluated based on a number of factors such as the time elapsed since its last use, the "stability" of its current level, the sign and magnitude of its most recent level change (negative changes are weighted more heavily), and its relevance to other concepts as determined by the number of branches of the tree connected to it. The highest scoring concept is selected for presentation to the student.

The student always has the option of vetoing this selection and choosing his own concept or accepting the system's second best choice. The selection process is described more formally in the Appendix.

PROBLEM GENERATION

After a concept has been decided upon, the system must generate a problem within this concept area unless the student prefers to supply one. The system attempts to tailor the problem difficulty to suit the student's level in that concept. Table IV gives some sample problems generated by the system.

As an example of the problem generating procedure, the algorithm used to form logical expressions for teaching about truth tables (see problem 3 in Table IV) will be presented in some detail. The basis of this algorithm is the probabilistic grammar shown in Table V which produces expressions utilizing binary operators with P, Q, R, S as variables. A probabilistic grammar is a formal language in which each rewrite rule is assigned a probability of being applied.

The first decision to be made is the number of variables in the final logical expression. As currently implemented, the probability of two variables is .5 for $0 \leq \text{level} \leq 1$ and decreases to 0 for level>1. The probability of 3 variables is .5 for $0 \leq \text{level} \leq 2$ and increases to 1 for level>2. The probability of 4 variables is .5 for $1 \leq \text{level} \leq 2$ and 0 elsewhere.

The incomplete logical expression is scanned from left to right for non-terminals. When the non-terminal symbol A is found (P_a+P_b) represents the probability of increasing the length of the expression where:

$$P_a(t)=P_b(t)=.75\,c/(n(t)+1.5c-1.5) \qquad (3)$$

where $n(t)$ is the current length and c is the number of variables in the expression $(c \geq 2)$

Since (P_a+P_b) is inversely proportional to the current length, the logical expressions do not become unwieldy. If the random number generated indicates that the

TABLE V

Probability	Rewrite rule
1:	$\mathfrak{D} \rightarrow A$
P_a:	$A \rightarrow (A^*A)$
P_b:	$A \rightarrow (\neg A)$
P_{c1}:	$A \rightarrow P$
P_{c2}:	$A \rightarrow Q$
P_{c3}:	$A \rightarrow R$
P_{c4}:	$A \rightarrow S$
P_d:	$* \rightarrow V$ (or)
P_d:	$* \rightarrow \Lambda$ (and)
P_e:	$* \rightarrow \uparrow$ (NAND)
P_e:	$* \rightarrow \downarrow$ (NOR)
P_e:	$* \rightarrow \oplus$ (exclusive or)

Constraints $0 \leq P_a, P_b, P_{c_i}, P_d, P_e \leq 1$
$P_a+P_b+\sum_i P_{ci}=1$
$2P_d+3P_e=1$

TABLE VI—Example of Student Interaction With System Executive

```
IS THIS THE FIRST TIME YOU HAVE USED CAI?
no
TYPE IN YOUR NAME
ebkoffman
AFTER REVIEWING YOUR RECORD, I HAVE SELECTED CONCEPT #9.
BINARY MULTIPLICATION
IS THIS OK?
yes
FORM THE PRODUCT OF THE BINARY NUMBERS 101.01, 1.001
IGNORING BINARY POINTS, WHAT IS THE PARTIAL PRODUCT DUE TO BIT 1 OF THE MULTIPLIER 1001?
10101
CORRECT
WHAT IS THE PARTIAL SUM SO FAR?
TYPE F TO GO FASTER    (Instead of answering the previous question, the student has hit the console attention button.
     S TO GO SLOWER    The system attempts to determine his wishes.)
     Q TO QUIT
f
WHAT IS THE PRODUCT OF 101.01, 1.001?    (The student's level has been increased by 1)
101.11101
CORRECT
YOUR LEVEL IN BINARY MULTIPLICATION INCREASED BY 1.2 to 2.3
I HAVE SELECTED CONCEPT #12
KARNAUGH MAPS
OK?
no
WOULD YOU LIKE TO PICK YOUR OWN CONCEPT?
yes
DO YOU KNOW THE CONCEPT NUMBER?
no
HERE ARE THE CONCEPTS CURRENTLY COVERED    (This printout would be omitted if the student had answered yes.)
 1. REGISTER 'ANDING'
     ⋮
15. STATE TABLE TO SEQUENTIAL DESIGN
WHICH CONCEPT NUMBER WOULD YOU LIKE?
15
FINE. THIS IS ABOVE YOUR CURRENT PLATEAU, BUT LET'S TRY IT.
DO YOU WANT ME TO GENERATE A PROBLEM?
no
HOW MANY STATES IN YOUR STATE TABLE?    (Student specifies problem parameters and works through solution)
5
HOW MANY INPUT BITS?
     ⋮
WOULD YOU LIKE TO CONTINUE PICKING CONCEPTS?
no
I HAVE SELECTED CONCEPT #12
KARNAUGH MAPS
OK?
yes
MINIMIZE A FUNCTION WHICH HAS 2, 5, 6, 7 AS MINTERMS USING THE KARNAUGH MAP METHOD
HERE IS THE KARNAUGH MAP    (A student with level <1 in this concept would be tested on his ability to construct the map)
     ⋮
```

expression should not be extended, one of the variables P, Q, R, S replaces A.

If the non-terminal symbol is *, one of the five binary operators is selected. P_e increases with level, while, P_d decreases. Hence, the more difficult operators are more likely in expressions generated for the students at higher levels of achievement.

Modifications of this technique are used to generate problems for combinational design and sequential design concepts. For the earlier number system manipulations, it is necessary to select a base for the number system and concatenate a set of valid digits from that base. As might be expected, the hexadecimal base appears more frequently at higher levels and the numbers to be manipulated are somewhat larger than those generated for lower levels.

IMPLEMENTATION

The system has been implemented on the IBM 360/65 at the University of Connecticut. It is programmed in the **C**onversational **P**rogramming **S**ystem (CPS).[5] CPS is a dialect of PL/I and includes some string processing features which have been extremely useful in programming this system.

There are forty IBM 2741 terminals connected to the main computer. CPS operates in low-speed core and each user is allowed a maximum of 12 pages of core (48 *K* bytes). Currently, space is reserved for fifty student records on disk. There is room for thirty concepts on the tree. There is an instructor mode which permits easy modification of the concept tree. Student records can also be printed out while in the instructor mode.

Students can log onto the system any time from 9 A.M. to 11 P.M. The average student session is about an hour in length. The student's record is automatically updated every 15 minutes so that the results of a complete session will not be lost in the event of a system failure.

The course coverage includes familiarizing the students with the binary, octal, and hexadecimal number systems. The students also learn how to use truth tables to represent logical functions. They learn techniques of combinational design and how to minimize the logic elements needed for a design problem. They are introduced to Flip-Flops and the design of sequential circuits. They also learn about machine-language programming by analyzing short program segments and indicating the changes incurred in various registers as these segments are executed.

Reaction from the students using the system has been very favorable to date. It has been used as a supplement to the regular lectures and homework assignments covering the material up through plateau four of the concept tree (Figure 2). The problem generation capability was not fully implemented which required students to insert their own problem parameters as they were requested by the problem solution algorithms. The major complaint of the students was the slowness of the system. This has been alleviated considerably now that problems are generated for all system concepts.

The system responds to a student input typically in 5 seconds or less. In a couple of the more complex solution algorithms (tabular minimization for example), a student working in the problem-solver mode may have to wait up to three minutes for a problem solution. Students at lower levels very rarely have to wait more than thirty seconds as the solution algorithms are executed in a piecemeal fashion. This, of course, could be improved substantially if compiled code (CPS is an interpretive system) and high-speed core were available.

During the current semester, a student group is using the CAI system in lieu of regular homework assignments. The complete tree of concepts is available. Table VI is an example of the dialogue that occurs between the system executive and the student. The system's messages are capitalized. All student inputs are preceded by a "—." Table VII presents examples of the interaction that takes place during a problem solution.

CONCLUSIONS AND FUTURE WORK

A generative system for teaching digital computer concepts has been described. This system includes an intelligent concept selection routine which attempts to individualize the path taken by each student through the tree of course concepts. In addition, the instruction and monitoring provided by each solution algorithm is dynamically adapted to suit a student's current level of knowledge. The system can also operate as a problem-solver and will provide portions of the solution process which have already been mastered by a student.

Expansion of the CAI system is continuing. The majority of the portion which teaches logical design has been completed. Students have also been exposed to an on-line simulator of the Digital Equipment Corporation PDP-8 which teaches machine-language programming.

Future efforts in this area will lead to a generative system which produces programming problems. As the problem is generated, a sequence of sub-tasks similar to a flow chart will also be produced. This will be used as the basis for teaching machine language programming and testing students' solutions. As before, portions of the program will be generated for the student and he will be required to supply the rest. The amount required from him will decrease as his level increases.

It is also anticipated that there will be further improvement of the concept selection algorithm. It is likely that performance can be improved by incorporating additional parameters into the algorithm as well as by adjusting the weights of the parameters currently used.

The techniques discussed in this paper could readily be applied to other courses which teach techniques of solving problems. In fact, several courses could be administered simultaneously by the concept selection routine. A concept tree would have to be specified for each course together with the algorithms necessary for generating and solving problems.

The initial burden on the course designer is, perhaps, greater than in a frame-oriented system as he must program general solution algorithms rather than a representative set of questions and anticipated replies.

It is possible that a "canned" solution program (such as ECAP for circuit analysis) could be converted into a teaching program by first segmenting it into course concepts. Each concept would be modified by inserting

TABLE VII

Example 1—Sequential Circuit Analysis

```
DERIVE THE TRANSITION TABLE FOR A SEQUENTIAL NETWORK WITH 2 FLIP FLOPS SUCH THAT:
  J1=(Y2 | X)    K1=(Y1 | Y2)    D2=(¬X)
AND THE OUTPUT Z=X & (¬Y2)
THE TRUTH TABLE IS SET UP AS FOLLOWS:
|     |   CURRENT     | CONTROLS |    NEXT      | CURRENT |
|     | INP STATE (t) |  At (t)  | STATE (t+1)  | OUTPUT  |
| ROW |   X  Y1  Y2   | J1 K1 D2 | Y1        Y2 |  Z(t)   |
|  0  |   0   0   0   |          |              |         |
TYPE IN J1
0
TYPE IN K1
1
NO. Y1=0,    Y2=0    SO(Y1 | Y2)=0
TYPE IN D2
1
TYPE IN THE NEXT STATE OF FLIP FLOP 1.
1
NO. IF J1=0 AND K1=0, THE FF STATE REMAINS THE SAME.
TYPE IN THE NEXT STATE OF FLIP FLOP 2.
1
TYPE IN Z(t).
0
|ROW| X  Y1  Y2 | J1  K1  D2 | Y1  Y2 | Z(t) |
| 0 | 0   0   0 |  0   0   1 |  0   1 |  0   |
| 1 | 0   0   1 |  1   1   1 |        |  0   |
(NOTE: BECAUSE LEVEL IS NOW >1, ADDITIONAL INFORMATION HAS BEEN PROVIDED FOR ROW 1)
TYPE IN THE NEXT STATE OF FLIP FLOP 1.
0
NO. IF J1=1 AND K1=1, THE FF CHANGES STATE.
TYPE IN THE NEXT STATE OF FLIP FLOP 2.
1
(NOTE: THIS LINE OF QUESTIONING CONTINUES UNTIL THE STUDENT'S LEVEL IN JK OR D FLIP FLOP
  ANALYSIS BECOMES ≥3 OR ROW 7 IS COMPLETED.)
  ⋮
ROW| X  Y1  Y2 | J1  K1  D2 | Y1  Y2 | Z(t) |
 7 | 1   1   1 |  1   1   0 |  0   0 |  0   |
NOW WE MUST FORM THE TRANSITION TABLE. THE HEADING FOLLOWS:
| CURRENT | CURRENT  |
|  STATE  |  INPUT   |
| (Y1, Y2)| X=0  X=1 |
WHAT IS THE NEXT STATE/CURRENT OUTPUT IF THE CURRENT STATE IS (0, 0) AND X=0?
(USE FORMAT (Y1(t+1), Y2(t+1))/Z(t))
(0, 0)/1
WRONG. Y2(t+1)=1
       Z(t)=0
IF CURRENT STATE=(0, 0) AND X=1?
(1, 0)/0
WRONG. Y2(t+1)=1
|(Y1, Y2)|  X=0     X=1  |
| (0, 0) | (0, 1)/0 (1, 1)/0 |
(NOTE: THIS LINE OF QUESTIONING WILL CONTINUE UNTIL THE PROBLEM IS COMPLETED
  OR THE STUDENT'S LEVEL BECOMES ≥3.)
```

TABLE VII—Continued

Example 1—Sequential Circuit Analysis

Example 2—Sequential Circuit Design

```
DERIVE THE STATE TABLE FOR A SEQUENTIAL CIRCUIT WITH A SINGLE INPUT LINE X.
THE OUTPUT WILL BE 1 WHENEVER THE LAST 4 VALUES OF X HAVE BEEN 1000

HOW MANY STATES ARE NEEDED?
5
WRONG. SINCE THERE ARE 4 BITS IN THE INPUT STRING, 4 STATES ARE NEEDED.
Q 0 IS THE STARTING STATE. QI CORRESPONDS TO THE FIRST I BITS BEING CORRECT.
Q 0 CORRESPONDS TO THE LAST INPUT 0 OR 1?
1
WRONG. BEING IN THE STARTING STATE MEANS THE DESIRED SEQUENCE HAS NOT BEGUN,

  SO THE LAST INPUT WAS 0
THE STATE IS Q 1 IF THE LAST INPUT WAS ____?
1
THE STATE IS Q 2 IF THE LAST 2 INPUTS WERE ____?
11
WRONG
Q 2 CORRESPONDS TO THE PARTIAL SEQUENCE 10
THE STATE IS Q 3 IF THE LAST 3 INPUTS WERE ____?
100

IN STATE Q 0: IF X=0 WHAT IS THE NEXT STATE/OUTPUT?
Q1/0
NO. THE LAST INPUT IS NOT PART OF A DESIRED PARTIAL SEQUENCE.
THE SEQUENTIAL CIRCUIT RETURNS TO THE INITIAL STATE Q 0
IF X=1?
Q1/0
IN STATE Q1: IF X=0?
Q2/0
IF X=1?
Q0/0
NO. THE LAST INPUT IS 1
THE NEXT STATE SHOULD BE Q1
IN STATE Q 2: IF X=0?
Q2/0
NO. THE LAST 3 INPUTS ARE 100
THE NEXT STATE SHOULD BE Q 3
IF X=1?
Q1/0
IN STATE Q 3: IF X=0?
Q0/0
NO. THE LAST 4 INPUTS ARE THE DESIRED SEQUENCE 1000
THE OUTPUT SHOULD BE 1
IF X=1?
Q1/0

THE FINAL STATE TABLE FOLLOWS:

|CURRENT|    INPUT     |
| STATE | X=0    X=1   |
|   Q0  | Q0/0   Q1/0  |
|   Q1  | Q2/0   Q1/0  |
|   Q2  | Q3/0   Q1/0  |
|   Q3  | Q0/1   Q1/0  |
|       |NEXT STATE    |
|       |  /OUTPUT     |
```

requests for student answers after the completion of each step in the solution process as well as remedial statements explaining the correct procedure to be followed for each step in the solution. In addition, a program statement would have to be included which would allow this question to be skipped by students above a specified level of proficiency.

It is felt that the extra effort required to program an algorithm based CAI system is worthwhile as the end product is a very versatile and flexible teaching tool.

APPENDIX—CONCEPT SELECTION

C_{ij} is the jth concept at Plateau i
L_{ij} is the level of C_{ij} where $0 \leq L_{ij} \leq 3$
n_i is the number of concepts at Plateau i
L_i is the average level of concepts at Plateau i, where

$$L_i = \frac{1}{n_i} \sum_{j=1}^{n_i} L_{ij}$$

If I is the current plateau, calculate L, where $L = I^{-1} \sum_{i=1}{}^{I} L_i$ and L is the average of all concepts at and below plateau I.

If $L \geq M$, where M is the Master average, then increase I by 1; otherwise, I does not change.

Now select the set of candidate concepts $K = \{k_1, k_2, \ldots, k_{nI}\}$ as follows:

Let $S_j = \{L_{xy} \mid (x \leq I) \Lambda (y \leq n_x) \Lambda (C_{xy}$ is a prerequisite of $C_{Ij})\}$ for $1 \leq j \leq n_I$

Calculate $\bar{S}_j$, the mean of S_j

If $\bar{S}_j \geq M$, then $k_j = C_{Ij}$, otherwise, $k_j = C_{ab}$ where C_{ab} is the prerequisite concept whose level, L_{ab}, is the smallest element in S_j.

Given the set of candidate concepts, K, the candidacy value, V_j, of each concept must be calculated. The concept, k_j, which has the highest candidacy value for this student is selected.

$$V_j = \sum_{i=1}^{5} w_i F_{ij}$$

$$F_{1j} = (2\Delta L_{Ij})^2 \qquad \text{(A1)}$$

$$w_1 = 1 \quad \text{if } \Delta L_{Ij} < 0$$

$$w_1 = 0 \quad \text{if } \Delta L_{Ij} \geq 0$$

Factor 1 tends to favor repetition of concepts for which previous usage led to a large negative change in level. The student is experiencing difficulty with this concept and its reuse with more interaction and monitoring occurring may prove beneficial

$$F_{2j} = \frac{(Q - Q_j)}{Q}, \qquad \text{(A2)}$$

where Q is the current sequence number and Q_j was the sequence number at the time concept k_j was last called. (Q increases by 1 each time a new concept is selected.)

$w_2 = 1$ if $Q_j > 0$

$w_2 = 3$ if $Q_j = 0$ (Indicates concept k_j has not been selected for this student.)

Factor 2 favors the recall of concepts which have not been used recently.

$$F_{3j} = R_j \qquad \text{(A3)}$$

where R_j is the stability of concept k_j as defined previously.

$$w_3 = -1$$

Factor 3 inhibits the use of a concept whose level is in a relatively stable range.

$$F_{4j} = \frac{|\Delta L_{Ij}|}{L_{Ij}} \qquad \text{(A4)}$$

$$w_4 = 1$$

Factor 4 is the percentage charge in level during the last usage of k_j. Factors 3 and 4 bias the selection toward concepts whose levels seem most likely to change during an interaction

$$F_{5j} = 2 \cdot D_j + E_j \qquad \text{(A5)}$$

$$w_5 = .1$$

Factor 5 calculates the number of prerequisites for concept k_j. Those prerequisite concepts which may be called as subroutines are counted twice. This factor gives emphasis to concepts which are more likely to result in the review of other concepts.

REFERENCES

1 J R CARBONELL
AI in CAI: An artificial intelligence approach to computer-assisted instruction
IEEE Transactions on Man-Machine Systems Vol MMS-11 No 4 Dec 1970 pp 181-189

2 J D WEXLER
Information networks in generative computer-assisted instruction
IEEE Transactions on Man-Machine Systems Vol MMS-11 No 4 Dec 1970 pp 190-202

3 M R QUILLIAN
The teachable language comprehender: A simulation program and theory of language
Communications of the ACM Vol 12 No 8 Aug 1969 pp 459-476

4 W R UTTAL et al
Generative computer-assisted instruction
Mental Health Research Institute Communication 243 1969 University of Michigan

5 IBM Corporation
Conversational programming system (CPS) terminal user's manual
IBM Report GH20-0758-0 1970

Preliminary thoughts about a UNIversal TEAching Machine (UNITEAM)

by JOE K. CLEMA, R. L. DIDDAY and M. WESSLER

Colorado State University
Fort Collins, Colorado

INTRODUCTION

Computer Aided Instruction (C. A. I.) promises to make practical the goal of enabling each student to receive highly individualized instruction at some point in his educational career. C. A. I. has evolved over the last fifteen years from programmed instruction which itself has had a relatively short history as an educational procedure.

Programmed instruction has not changed dramatically since the time of Pressey,[1] who in 1926 used it as a means of reinforcement in the educational process. Thirty years later Skinner[2] saw the need for incorporating stimulus materials into a framework of instructional aids in order to provide education without a live teacher. However, the ultimate goal of a universal teaching machine which provides instruction to individuals in arbitrary fields has not yet been achieved.

We consider a universal teaching machine to be one which may be converted to teach any subject by simply changing the program data base and which can adapt the instruction to the needs of each individual student. It is desirable that the transformation from the natural language of the subject to the machine language of the data base be somehow simple.

TEACHING MACHINES AS QUESTION-ANSWERERS

For us the problems faced by a general teaching machine are similar to those faced by question-answering systems. A question-answering system is one which accepts information and uses it to answer questions. Question-answering systems are attempts to construct and manipulate data bases which are represented as associations among symbols which have meaning when humans see them. Newer systems are based on theorem provers,[3] and when given some input question must choose efficiently from an infinity of possible deductions in order to arrive at an answer. At present the limiting factor in their usefulness is the difficulty in selecting an appropriate sequence of deductions in a reasonable length of time.

A question-answering system does not store as such all information which is available to the user. Instead, it maintains a compressed data base of compactly coded facts.[3,4,5] The facts may reside in data structures[3,4,6] such as lists or binary trees. There are many methods for deducing answers which have not been explicitly stored in memory. These include:

1. A set of prewritten programs, one for each class of question. These programs are a permanent part of the system and do not change from their initial state.
2. A translator which creates a subprogram each time a question is input to the system. This subprogram exists only as long as it is needed to answer the question.
3. A formal theorem prover using some subset of current known facts as axioms.

The problem with prewritten subprograms is when questions arise that require interaction among existing classes. Either the interaction must be anticipated or new subprograms must be written. In No. 3 the inference mechanism must be general enough to answer questions of a variety of types.

An important characteristic of all question-answering systems is the set of languages programmed into them. At the most sophisticated level are the dialogue languages[3] which include:

1. A language for introducing facts into the system;
2. A query language to be employed by the user;
3. An answer language to be employed by the system.

In the process of answering questions the system may require additional information from the user. Thus it is necessary that a query language be employed by the user and an answer language be employed by the system.[7]

Question-answering systems also consist of one or more internal languages. These are machine languages which are used as intermediate steps in translation[8] or for calculation and manipulation of data.

The problem of representing data may be derived into three parts:[7,10]

1. Determining the semantic content of the data;
2. Choosing a language to represent the semantic content;
3. Choosing an internal representation of the language chosen above.

For example, the sentence "GASP is FØRTRAN based." may be expressed by the binary relation "is" as applied to the subject "GASP" and the predicate "FØRTRAN based." In choosing a language to express semantic content we might use the notation of symbolic logic, and in choosing an internal representation we might express a series of binary operations as a binary tree structure.[9,11]

OUR APPROACH

A teaching program can be structured so that the burden of finding a path through the data, i.e., from question to answer, is placed on the student. To put it another way, we can generate proofs at random. Of course we will not know what theorem we have proven (what question we have answered) until we are through, but this seems to be the way that human teachers operate anyway.

Currently we plan to represent a topic as an ordered set of information units called *concepts.* A concept consists of an ordered triple (a production), a string of symbols called the *question format,* a list of concept labels called the *error list,* and an integer weight which serves as a measure of how well the student knows the concept.

In operation the system will be able to access a subset of concepts, namely those it has introduced so far. Associated with each student is a vector, the elements of which serve as the *weights* of each concept in the topic when he is using the system. When the student is asked a question, the system selects a concept with a probability skewed by his weight vector.[13,14,15,16] This involves Monte Carlo methods and techniques for generating pseudorandom numbers. A certain number of deductions are then made, again chosen according to the weighted distribution. As each deduction is performed, the question format of each successive concept is taken into account so as to generate a reasonably well-stated question. At some point in the random deduction process a deduction is not made according to the production associated with the current concept. Instead, an alternate concept is chosen from the error list, and its production is carried out next. This allows the use of "what is wrong with . . .?" type questions which we will demonstrate later.

When a student's error vector fits certain criteria, a new concept will then be introduced and used (with high probability) in the question generation process. It is our basic assumption that if a student can answer arbitrary questions which result from associations among a basic collection of concepts, he has learned the topic. It is in this respect that we call our programs *teaching programs.*

Where do concepts come from? Unlike the designers of question-answering systems, we make no effort to find an elegant or concise representation. We believe that concepts should be the embodiment of a global model of the topic encoded in "symbology" which is as close to the natural language as possible. We further envision that such a program could be developed in the following way.

Experts in a field such as mathematics, programming, etc., will be engaged to extract from their topic a set of concepts which describe it. By "describe" we mean that the concepts they provide can potentially lead to the derivation of all facts that they are trying to teach in their subject area. We can think of the topic as a vector space and the concepts as a set of basic vectors which describe it. We imagine that the process of extraction will be done on-line, i.e., at any point the designer can interrogate the concepts he has previously defined in order to see what, in fact, can be derived from them. Thus he can refine concepts which lead to false deductions.

It has been implicitly assumed that we can embody everything necessary about a topic within a reasonable number of concepts. Also, the usefulness of this technique depends on its efficient implementation so that a large number of students can interact with the program simultaneously and economically. The former requirement dictates that the productions associated with each concept be potentially powerful. The latter requirement dictates that they be easily manipulated. A similar problem is faced by those who write the software for computer languages. ALGOL is represented as a "context-free" (Chomsky Type 2) set of productions, as opposed to Chomsky Type 1 which is "context sensitive," which are easy to manipulate but which do not completely specify the language. Context sensitive aspects of the language such as the requirement that array sizes be explicitly declared have been shelved away in symbol manipulation routines, etc.

This solution is one we cannot take. Our goal is to expose all aspects of the subject within the framework of these concepts. We apparently will be forced to use a context sensitive scheme in every interesting case.

COMPLETED WORK

In order to test our approach, we have chosen to design a program, UNITEAM, which teaches the simple programming language BASIC. We have further simplified our immediate goal, choosing to divide the teaching into three phases. We hope that those phases can eventually be included in one general framework which will be applied to a broad range of topics. Phase 1 we term "motivational" in that we are trying to relate the new words "computer", "memory", "word", "I/O", etc., both to things with which the student is already familiar and to each other.

In Phase 2 the syntax of the language and its semantics with respect to the concepts of Phase 1 are introduced. In Phase 3 the semantics in terms of the world and more global techniques of programming are emphasized. The following reports our work in Phase 2, mentioning our approach to the other two phases only in passing.

Sample productions belonging to concepts which will be stressed in Phase 1 are things like:

COMPUTER HAS-AS-PART MEMORY
COMPUTER HAS-AS-PART CONTROLLER
COMPUTER HAS-FUNCTION PERFORM
  OPERATION
etc.

Sample questions might be something like:

WHAT PART OF COMPUTER HAS-
  FUNCTION PERFORM OPERATION?
etc.

There is much work to be done on this phase which is the most general phase. For example, we must understand what effect allowing various verbs (second element of the production) has on the system. In Phase 2 we deal almost exclusively with productions which have the same verb.

In Phase 2 we are currently using techniques similar to those used in compilers for languages expressed as context free grammars (Chomsky Type 2). Since there are also global (context sensitive) features, we make a slight addition. A typical production is

/line/ IS UNIQUE (/statement number/0/
  statement/CR

which would appear in traditional Backus-Naur form[17] as

/line/ ::= /statement number/statement/CR,

and says that a /line/ consists of a /statement number/, a /statement/, and a carriage return. It is a global feature not expressed in the second form above that no two lines may have the same statement number. In our system the expression "UNIQUE( )" will reference a subprogram which insures that any statement number that is generated through this production is unique. There will be an additional concept with the production

UNIQUE( ) HAS-FUNCTION EACH LINE
  HAS A UNIQUE /statement number/

which summarizes the operation of the hidden subprogram for the student.

A second difference is that, unlike compilers, we will proceed from left to right which requires the rewriting of a few productions from their more commonly seen forms.

With this brief introduction let us sketch the operation of the system as it would proceed in generating a question. For this example we will generate a statement as it would appear in BASIC[18] except for one error. We will not write all the productions which are necessary, but suppose that concept 23 has been chosen. Concept 23 looks like this:

C_{23}→./assignment/ IS LET / variable /
  = / arithmetic expression /
  .QUESTION FORMAT = ...
  .ERROR LIST = $\wedge$ (null)
  .WEIGHT = ...

We will have three push-down stacks:[19,20] One stores the concepts in order of application, one stores that part of the final string we have completed, and one stores the parts we are still working on. We call these the C stack, T stack, and W stack respectively.

So far the stacks look like this:

C Stack	W Stack	T Stack
C_{23}	LET	LET
⋮	/variable/	⋮
	=	
	/arithmetic expression/	
	⋮	

Next we choose a concept which has a production that tells us something about the top-most (non-terminal) element of the W stack. Suppose we choose

C_{35}→./variable/ IS / identifier/

.

.

.etc.

Then the stacks would look like:

C Stack	W Stack	T Stack
C_{35}	/identifier/	LET
C_{23}	=	⋮
⋮	/arithmetic expression/	
	⋮	

At some point in the processing the student will encounter a concept which has a relatively high weight, meaning that the student has had trouble with it before. Instead of copying the right side of that concept's production onto the W stack, the program copies the right side of a production belonging to one of the concepts in the current concept's error list. This fact is noted on the C stack.

Suppose that later in the processing the top of the W stack contained/arithmetic primary/, and instead of an appropriate match we choose from the appropriate error list

C_{24}→/logical expression/ IS /arithmetic primary/ /relation/ /arithmetic primary/.

The stacks would end up looking like this:

C Stack	W Stack	T Stack
C_{24} ERROR	/arithmetic primary/	=
C_{28}	/relation/	B4
C_{25}	/arithmetic primary/	LET
C_{37}	⋮	
C_{35}		
C_{23}		

Imagine that finally we had:

C Stack	W Stack	T Stack
C_{35}		7
C_{31}		<
C_{41}	⋮	A2
C_{35}		=
C_{31}		B4
C_{24} ERROR		LET
C_{28}		⋮
C_{25}		
C_{37}		
C_{35}		
C_{23}		

The system prints

WHAT IS WRONG WITH THIS /assignment/?
LET B40 = A20 < 7.

Note that /assignment/ is the left side of the production of the bottom concept in the C Stack (C_{23}).*

If the student's answer looks something like

/logical expresion/ SHOULD HAVE BEEN / arithmetic expression/,

i.e., if it names expressions which pinpoint where the error was made, the program accepts it. If the student's answer includes expressions found *higher* in the C Stack than the ERROR flag, for example, /relation/ SHOULDN'T BE THERE, the program responds:

YES, BUT CAN YOU BE MORE SPECIFIC?

Otherwise the program limits the productions which caused the error and any others in the C Stack that the student might want to see.

We have not yet settled on the best way to locate an error nor on the best way to update the weights of those concepts used in generating a question.

Phase 3 is dedicated to teaching the student how to write small, meaningful programs. It will begin with concepts which describe how the various BASIC statements alter the variables referenced within them. The next batch of concepts will introduce bits of programs such as "short routine", "sum an array", etc. Finally, the highest concepts will be oriented toward a canonical representation of a few simple programs. Thus, if all goes well, at the end of his relationship with the system the student will be asked questions about what specific, complete programs do.

CONCLUSION

We have outlined our initial approach to the problem of creating a universal teaching program. The fundamental ideas are similar to the theorem proving approach, i.e., the technique used by question-answering systems, but runs "backwards," generating questions which test the student's ability to manipulate the basic concepts of the topic he is to learn. We have given an example which, while extremely simplified from the general case, demonstrates a sophisticated capability when compared to present day teaching programs. It remains to be seen if the techniques we devise can be generalized sufficiently to accept concepts derived from diverse topics.

The parameters used to evaluate such a system are access time, CP (Central Processor) time, IØ (Input/Output) time for Control Data Corporation 6000 machines, mass storage, and core requirements. We believe that all four can be cut drastically by utilizing the random access capability of the Control Data Corporation 6400 on which our program is being implemented. That is to say that the system will be maintained on disc. Since the selection of concepts is inherently random, this approach seems intuitively correct. The elimination of chaining, i.e., searching for data sequentially, will minimize access time, and CP time will be reduced by holding data manipulation to an absolute minimum. Since the system will reside on mass storage units, core capability will be greatly enhanced.

The principal advantage of UNITEAM over other systems (PLANIT, PLATO, ETC.) is that UNITEAM is the most adaptive program yet devised. Certainly the ultimate C. A. I. will include verbal communication and pictures, as does PLATO. PLANIT can recognize responses in a number of ways which UNITEAM at its present stage of development cannot.

PLANIT can derive data from several sources in order to provide a basis for making instructional decisions. This examination of various sources of data and records is incorporated into UNITEAM to formulate the weights (UTILES) of the value of various topics to be presented to the student.

1. Student performance on a question;
2. Cumulative student performance;
3. Student entry characteristics;
4. Student preference.

Student performance includes accuracy of response, time of response, etc. Student entry characteristics include the ability and educational background. It is felt that UNITEAM should make use of the prior knowledge of an individual, including such things as grades, IQ, interests, teacher ratings, etc.

Feedback is a major factor in determining which material and at what depth that material should be presented next, thus enabling the program to adapt to the needs of the individual. All previous feedback including the most recent information in one sense is reevaluated at each step since UNITEAM is based on stochastic methods. Although UNITEAM bears some Markov chain similarities, it is more complex, and thus final evaluation will rest with performance of students rather than mathematical analysis. Thus, total examination of feedback determines to a large degree which direction one next proceeds.

* The concept numbers correspond to a list we have used but which is not included in this paper.

At the present time feedback consists of:

1. Student records;
2. Evaluation of performance;
3. Number of prompts, hints, cues needed;
4. Elaboration on a student's response as a means of reinforcing a concept.

Unlike other C. A. I., UNITEAM employs Monte Carlo methods in addition to a weighting scheme associated with topics, concepts, etc. This system provides a completely non-linear capability and innumerable paths of deductions as UNITEAM develops necessary logic at each stage or step in the question-answering process. UNITEAM is capable of providing an almost unlimited number of pathways through the instruction because its responses are dependent upon each individual student who is making his own decisions, and it is very unlikely that any two students would choose the same path through UNITEAM.

APPLICATION OF UNITEAM

The system design of UNITEAM envisions the following to be provided by the instructor:

1. Text or subject material (data) to be punched on cards;
2. List of key words which students are to know;
3. Table of weights (utiles) for various topics and concepts;
4. Table of yes-no and true-false questions.

Instructors are not required to know anything about programming or how UNITEAM works. They must know how to present material, and if they wish, UNITEAM will handle the weights of all topics and concepts equally by default. The table of key words provides ready access to the determination of important concepts.

At the current stage of UNITEAM's development, response time is not a problem. This, of course, is because at its present stage of development, the question-answering is one of essentially true-false or yes-no. Development is centering on multiple choice responses, and definitions at this time, and the response time problem is appreciated as the question-answering becomes of a more complex nature. Disc storage rapidly becomes a problem, as we envision the capability of UNITEAM to essentially control access to all knowledge about a particular course being taught that an instructor might like to include. For the controlling system (UNITEAM) disc storage is not a problem, but the course material (data) rapidly can become unwieldy. With the development of such hardware as Precision Instrument Company's trillion-bit laser mass memory, one might hope that this problem will soon be eliminated or at least greatly reduced.

REFERENCES

1 S L PRESSEY
A simple apparatus which gives tests and scores—and teaches
In School and Soc Vol 23 1926 pp 773-776

2 B F SKINNER
Teaching machine
In Science Vol 128 1958 pp 969-977

3 C C GREEN
The application of theorem proving to question-answering systems
Information Service 1970

4 J R SLAGLE
Experiments with a deductive, question-answering program
In Comm ACM Volume 8 1965 pp 792-798

5 J D BECKER
An information processing model of intermediate level cognition
In Stanford Artificial Intelligence Project Memo AI-119

6 D E KNUTH
The art of computer programming—Volume I—Fundamental algorithms
Chapter 2 Addison-Wesley Publishing Company Reading Massachusetts 1968

7 C C GREEN
The application of theorem proving to question-answering systems
Information Service 1970

8 F R A HOPGOOD
Compiling techniques
American Elsevier Publishing Company Inc New York 1969 pp 29-34

9 D E KNUTH
The art of computer programming—Volume I—Fundamental algorithms
Chapter 2 Addison-Wesley Publishing Company Reading Massachusetts 1968

10 J R SLAGLE
Experiments with a deductive, question-answering program
In Comm ACM Volume 8 1965 pp 792-798

11 D MAURER
Introduction to programming
Stanford University Press 1967

12 F R A HOPGOOD
Compiling techniques
American Elsevier Publishing Company Inc New York 1969 pp 29-34

13 T H NAYLOR J L BALINTFY D S BURDICK K CHU
Computer simulation techniques
John Wiley and Sons Inc New York 1966 Chapters 3 and 4

14 J M HAMMERSLEY D C HANDSCOMB
Monte Carlo methods
John Wiley and Sons Inc New York 1964

15 *PLANIT author's manual*
Systems Development Corporation Sections I and II 1971

16 A D CALVIN
Programmed instruction
Indiana University Press Bloomington Indiana 1969

17 P NAUR et al
Revised report on the algorithmic language ALGOL 60
In CACM Volume 6 page 1 1963

18 J G KEMENY T E KURTZ
Basic programming
John Wiley and Sons Inc New York 1960

19 F R A HOPGOOD
Compiling techniques
American Elsevier Publishing Company Inc New York 1969 pp 4-10 52-54

20 D E KNUTH
The art of computer programming—Volume I—Fundamental algorithms
Addison-Wesley Publishing Company Reading Massachusetts 1968 pp 234-238

21 *PLANIT author's manual*
Systems Development Corporation 1971 Sections I and II

Mainline CAI, necessary but not oppressive*

by C. VICTOR BUNDERSON

University of Texas
Austin, Texas

Two years ago the term "mainline instruction" was introduced as a referent to a carefully designed, total instructional system to replace complete courses with a far less labor-intensive mixture of men and machines.[2,3] This mode was contrasted with the adjunctive use of the computer by faculty members who are happy to work within the system and merely want to use the computer to improve their instruction. The two NSF-sponsored conferences on Computers in the Undergraduate Curriculum[11,6] have illustrated and stimulated the adjunctive approach.

The term "mainline instruction" is not fully descriptive of the concept it designates. It serves as a shorthand term for *individualized instructional systems of major scope, using the computer as a tool for instructional management and information transmission, developed according to a design science approach and used as a less costly alternative to conventional instruction.* The ends to which computer uses are directed in this mode are necessarily controversial for: (1) mainline instructional systems revise the role of the teacher substantially, (2) mainline systems imply the emergence of a new profession of "courseware design" backed by instructional scientists who develop the instructional theorems used by the courseware designers, and (3) it is feared by some that the transformation of education from labor-intensive to technology-intensive systems might be accomplished only at the sacrifice of certain values of our liberal education tradition.

Advocates of the adjunctive use of computers in education point out that their approach bypasses these controversial aspects of the more radical mainline approach. By leaving the teacher firmly in control they minimize the threat to members of that profession, and more easily avoid charges of "dehumanization." By pointing out that no generally accepted theory of instruction exists, nor ready translations of learning psychology to instruction, they keep the teacher in control of program development and put off the problem of standards for evaluating programs. Because of the merits of the adjunctive mode in opening instruction to more student-oriented, problem-solving approaches, they can argue that adjunctive computer use in instruction not only preserves important values of our liberal tradition, but enhances them.

Arthur Luehrmann has been one of the more articulate and positive advocates of the adjunctive approach. Among the values to be sought through this mode are the following:

(1) to bring about a qualitative restructuring of the curriculum toward algorithmic and problem-solving approaches, and
(2) to encourage independent, self-motivated learning, and investigation using the computer as a tool.[7]

A third value should be added to this list, for Luehrmann and his colleagues at Dartmouth have been leaders in this effort for years—

(3) to bring about widespread and flexible access to computer use in higher education nationally consonant with the recommendations of the Pierce report.[8]

My intention in writing this paper is not to challenge these values, for I share them. It is not to argue that the adjunctive approach is not a good way to seek the three goals listed above, for I believe it is and that it will be the more visible way to achieve them for the next few years, touching many more students than will mainline CAI. My purpose in this paper rather is to present data and rationale supporting the propositions that:

(1) mainline systems are necessary for the widespread dissemination of computers throughout the educational establishment; and

* Research reported in this paper has been supported by NSF Grant GJ 509 X. TICCIT courseware development is supported by NSF Grant GJ 28095 and a MITRE Corporation subcontract.

(2) that contrary to stereotypes built up through comparison with earlier attempts at tutorial CAI, mainline systems can lead to the first two goals listed above and need not detract from other important values of liberal education.

THAT MAINLINE SYSTEMS ARE NECESSARY

The Pierce report recommends that steps be taken to provide all college and university students needing such services with adequate computing services. It does not specify how this is to be done. A direct investment of the Federal Government to support the free and relatively unstructured use of computers in the adjunct mode would represent a staggering capital investment. The government has not risen to this challenge. To examine some reasons for their reluctance to do so, the need for computers must be put in context with other problems being faced by education in this country which also demand additional capital. Some of these problems are accompanied by compelling arguments for higher priority than the Pierce report recommendations.

Colleges and universities are facing serious financial problems and social pressures which call for rapid and dramatic change. Consider the following facts. During the past ten years, the gross national product increased 93 percent. During the same period, the cost of higher education increased 266 percent. An ever-increasing percentage of this rise was attributable to faculty salaries. Instructional outlays accounted for 45 percent of the costs of higher education in 1945. In 1971, they accounted for 65 percent. Approximately 90 percent of instructional costs are attributable to faculty salaries.

While salaries have been increasing, productivity, as expressed by teacher-pupil ratios, has been decreasing. In 1959-60, the ratio was one teacher to 26 students in higher education. In 1969-70, it was 1 to 20.

In an analysis of population trends for the 100 largest standard metropolitan areas, the Academy for Educational Development[10] projected that the population of 18-24 year olds will increase from 12.5 million to 15.7 million between 1968 and 1990 (26 percent). During this same period of time, however, collegiate level enrollment was projected to increase from 4.1 to 8.2 million, or 100 percent. This increase reflects increasing numbers of disadvantaged youth entering college, increases in married women returning to school, more employees upgrading skills, more technical and vocational education, and more leisure time due to automation. The cost estimate for this greatly increased body of students is $24 billion in 1990, up 200 percent from the $8 billion cost in 1968. This estimate covers operating costs alone. It does not reflect capital construction nor athletics.

Legislatures, state and federal, are refusing to support these trends. In an attempt to help beleaguered college presidents meet the resulting cost squeeze, the Academy for Educational Development listed 115 ways to decrease the costs of higher education. First on the list was reducing the number of non-tenured faculty, adjunct or part-time faculty, consultants, and research and teaching assistants. Another suggestion was to cut computer costs or postpone the acquisition of new hardware. However, if CAI could be shown to be a cost-effective alternative to more labor-intensive methods, college presidents would have many more options, including, in some cases, the option to increase enrollment and effectiveness simultaneously.

Clearly, radically different approaches to education are needed. It will not be possible economically to continue the teacher/classroom model indefinitely as the prime method in higher education. Peter Drucker[4] has expressed this thought as follows:

> *The educators still talk of minor changes, of adjustments and improvements. Few of them see much reason for radical changes. Yet education will in all likelihood be transformed within the next decades by giant forces from without.*
>
> *It will be changed, first, because it is headed straight into a major economic crisis. It is not that we cannot afford the high costs of education; we cannot afford its low productivity. We must get results from the tremendous investment we are making . . . .*
>
> *Teaching is where agriculture was around 1750, when it took some 200 men on the farm to feed one nonfarmer in the town. We have to make the teacher more productive, have to multiply his impact, have to increase greatly the harvest from his or her skill, knowledge, dedication, and effort. Otherwise we shall run out of teachers—even if we do not run out of money for education.*

It is difficult to see how adjunctive CAI can meet the requirements implied by Drucker for educational reform. The philosophy of adjunctive CAI accepts without question the central role of the classroom teacher. Materials are prepared with teachers in mind to provide the context and motivation for the usage of computer packages. This assures that computer use will represent an add-on cost to existing costs for labor, overhead, and physical plant. It also makes it difficult to disseminate the package widely, for a teacher training project of some kind is necessary to enable the new user to adapt the inductive approach to instruction usually implied, provide the necessary context and motivation, and operate the program skillfully.

Because of its systematic development and "total system" approach, mainline CAI systems have by design provision for context, motivation, sequencing and correct utilization of each system component. Training programs are obviously required for the teachers and others who will still be involved in the system, but with a mainline system the student to teacher ratio is greatly increased, and the costs of the development and operation of retraining programs are spread over a much broader base.

These factors lead any economist to predict much greater potential for mainline CAI than for adjunctive to produce mass dissemination of computers in education, assuming the demonstrated solution to problems of hardware, software, and educational effectiveness. Wide availability of multi-terminal computer systems for administering these programs, if properly configured, will also provide a resource for enormously expanded activity of the adjunctive variety. Minicomputer systems designed for mainline courseware could also provide computer power for running small student jobs, e.g., in BASIC. They would have to be tied to larger systems to permit large data base simulations and other powerful adjunctive applications. It is most important that students and teachers have access to adjunctive computer resources, for such activity does lead to qualitative improvement of aspects of the curriculum; creative ideas which can be implemented more broadly, both by packaging within mainline systems and by easy transfer to compatible systems.

ENHANCING IMPORTANT EDUCATIONAL VALUES IN THE DESIGN OF MAINLINE INSTRUCTION

Early attempts at tutorial CAI, or other forms of programmed, packaged instruction designed to carry a substantial burden, have been criticized for being plodding, inflexible, unimaginative, and stultifying to the intellectually able and creative student. Too often these criticisms have been justified. But to judge the potential of any system involving considerable tutorial CAI by these examples is equivalent to judging the potential of motion pictures, having never seen a modern movie, on the basis of a few amateur home movies.

Consider the two values stressed by Luehrmann.[7] The first is that it is desirable to bring about a qualitative restructuring of the curriculum through computer use. The use of algorithmic and information-processing approaches to representing and illustrating concepts in new and powerful ways is an important contribution of the computer. Programs which make such a contribution can be used either by a teacher in the adjunctive mode or within a mainline system. In addition to this, the systematic analysis of a block of knowledge in mainline program design can lead to another form of qualitative restructuring, also powerful and effective in its educational consequences. This systematic behavioral analysis forms an essential part of any large-scale courseware development project. It involves the specification of measurable performance objectives in relation to broader course goals and the construction of a "learning hierarchy" which relates the major objectives and enabling objectives to one another by means of a kind of flowchart which shows prerequisite relationships and other dependencies. Such an analysis usually cuts away numerous topics which for years have been taught because of tradition rather than instructional merit. It simplifies and streamlines the structure of a course and acts as a theory for the steps a novice must take to attain the performance capabilities of a competent graduate. Such an analysis may reveal fuzzy concepts and suggest new research. It results in a representation of the course of study and its criteria for success which is more comprehensible to both students and teachers than outlines or syllabi.

An example of the effectiveness of this approach is found in an analysis of that portion of beginning Arabic dealing with the writing and sound system.[1] In class it takes 6 weeks, 6 periods per week to teach this complex non-Latin script and its phonetics. Our CAI program takes students from 5 to 9 hours at the terminal, augmented by 4 hours with an instructor. Comparison studies have shown that the students' performance surpasses by far that of groups learning in classroom or language laboratory. Only a part of this gain can logically be attributed to the computer. The behavioral analysis and specification of appropriate practice for each objective was the key, and classroom work could be greatly enhanced by its adoption regardless of the existence of a computer.

Another goal discussed by Luehrmann[7] was to encourage independent, self-motivated learning and investigation using the computer as a tool. It is quite as possible to seek this goal in the design of mainline courseware as through adjunctive approaches. The National Science Foundation has funded two major demonstrations of computer-assisted instruction: the PLATO system, under development by The University of Illinois, and the TICCIT system, under development by the MITRE Corporation. Under subcontract to the MITRE Corporation, a group of instructional psychologists at The University of Texas and at Brigham

Young University are developing the courseware for the TICCIT system. This courseware may be described as a set of mainline systems in the areas of freshman mathematics, freshman English composition, remedial English, and remedial mathematics. The target is the junior colleges of this nation. The four mainline systems are designed to replace up to 20 percent of the classroom instruction in the typical junior colleges at less than $1.00 per student hour (instead of the existing $1.50 for instruction, or $3.20 for total costs now extant).

In the development of courseware for the TICCIT junior college project, we are designing novel systems which will enable students to achieve on each of four major kinds of objectives, each measured by special scoring systems implemented in the courseware. These are:

(1) All students will achieve mastery at either A, B, or C level (student contracts for the level) on the lesson and unit tests that make up the measurement system for mastery.

(2) All students will develop a more positive attitude toward the subject matter and mode of instruction as measured by increases in their tendencies to approach voluntarily optional work related to the subject matter. More precisely, we provide opportunities within the course for optional work, use a variety of techniques to encourage approach responses, and measure our success. In some ways, we feel that approach is the master criterion, for it determines the student's future relationships with the subject matter. We can teach him to hate it and avoid it in the future or to like it and approach it in the future. This criterion seems especially important at the junior college level.

(3) All students should develop improved strategies for learning relative to the subject matter and the TICCIT system implementation. Strategies are operationally defined in our courseware as a vector of integers referring to items on one or more menus presented to the student. The order of the integers describes a sequence through the items on a particular menu. Menus exist at the course level, unit level, and lesson level, and serve as hierarchically organized points of control. A strategy may be entered in any of two different modes: survey or instruction. Provision for review is also available.

We will measure our success in improving strategies by changes in efficiency scores (mastery ÷ time) and by reference to strategies found to be effective for certain outcomes by certain classes of students, especially the more effective students. A high-level advisor program will exist to encourage and help students in the selection of good strategies.

(4) We wish students to develop a sense of responsibility for their own learning. We want them to feel that education is their choice and that both the positive and negative consequences of it are to be chosen or rejected by them alone. What we wish to convey to the student is that he is not just the recipient of orders but a participant in decisions that shape his life. He will be given the opportunity to choose a grade level, try various strategies, choose to take optional extra work or not, and choose among various "fun options" (games, films, computation and plotting programs, etc.). From the instructional system and the humans who manage it, the student should get the clear message:

> "We provide effective, efficient, and palatable instructional resources and a way of measuring your progress toward your goals. Choosing among alternate goals is your responsibility. Working effectively is your responsibility. If you fall below your potential we provide sympathy and advice. If necessary, we provide automatic control for a time, always encouraging the independence that assures you of control over your own destiny."

Measurement of changes in responsible behavior in the use of learner control options and commands is a complex process and will initially be done only at a gross level. These gross scores will locate the student somewhere between yielding control completely to the program and permitting him to push control buttons wildly or to jump about within the course without making progress on mastery and efficiency.

Such an ambitious scheme can only be attempted within the context of large mainline instructional systems. We estimate that the average student will sit at the terminal for at least 50 hours for five hours of semester credit. The mechanisms we have and are designing to achieve each of these objectives cannot be described in this short paper, but some flavor can be given. Briefly, we seek to achieve these objectives through three major sets of novel approaches which have never been used before, to our knowledge, in any CAI program. These are: (1) new control structures, to facilitate both learner control with program control options, (2) the construction of a high-level "advisor program," (3) the use of a variety of motivational techniques.

CONTROL STRUCTURES AND LEARNER CONTROL

Three years ago The University of Texas CAI Laboratory developed a modestly sized mainline CAI system to teach mathematics prerequisite to freshman science courses.[9] This program employed learner control of sequence, amount of practice, and testing very extensively. Hierarchical index structures, menus, and a separate control program were implemented. A series of research studies was conducted to analyze the effects of various control options (e.g., Judd, Bunderson, and Bessent).[5] These studies were inconclusive regarding the effects of learner control vs. program control on mastery and speed criteria. Though equivocal, the results indicated that little might be lost through the use of learner control in CAI on mastery and speed. It later occurred to us that much might be gained, and that learner control was more likely to be related to measures of approach, strategy, and responsibility than to mastery. Gambling that evaluation will prove us right, but that if it doesn't the program control options can take over, the control structures developed in the MathS course were revised and improved, and are being implemented in the TICCIT courseware.

Methods of control differ somewhat from course to course and among levels within courses. A number of the important concepts are illustrated in Figures 1 and 2. Figure 1 represents the components of a lesson menu from the precalculus mathematics course. The objectives section tells the students about the structure of the lesson. Those students who opt for this material learn that there are two major supporting objectives, corresponding to the two instructional sequences (items 5 and 6). Section 5 provides structured instruction to enable the student to achieve the objective: Given a second degree polynomial function in standard form, perfect square form, and factored form, identify and plot the extreme point, zeros (if any) and y intercept. Section 6 teaches algorithms for transforming functions from one form to another, and for finding zeros. Within both of these instructional sequences various control options and paradigms derived from recent research and theory in instructional psychology are used. The "Review Tips" section discusses the prerequisites (concept of zero, domain, range, function, etc.), provides brief review material, and references earlier lessons. The "So what?" file contains information on why a junior college student should bother to learn these objectives. The mini-lesson is a quick survey of the entire lesson. Definitions are provided for the important concepts in the lesson, including second degree polynomial and the three forms of the function. The mastery test is diagnostic. If the student fails parts of it he is given advice which is designed to help him go back to the lesson menu, or to review, before taking the test another time.

Lesson Menu

Second Degree Polynomial Functions

1. Objectives
2. Review tips
3. So what?
4. Mini-lesson
5. Definitions
6. Instruction: zeros, extreme point, y intercept, graphing
7. Instruction: algorithms
8. Mastery test

Figure 1—Sample menu for lesson on second degree polynomial functions

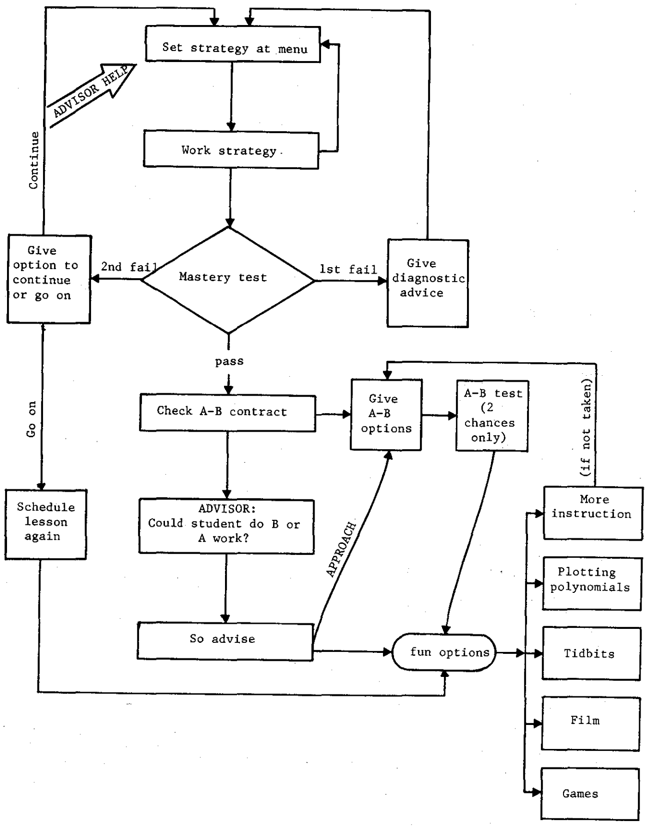


Figure 2—Generalized flow within a lesson

Figure 2 is a nondeterministic flowchart in which the overall pattern for most lessons is revealed. Students may try several strategies before taking the mastery test, and after failing it the first time. The advisor program may be called by the student for a discussion of various strategies used by fast, slow, and average students, or the advisor program may interject commentary under certain conditions.

A student is not required to attempt a mastery test more than twice in close succesion, but may elect to come back to the lesson at a later time. Under this condition the lesson is automatically rescheduled. The student may request "status" from the advisor program at any time to see what lessons he has completed.

Once the mastery test has been passed, if there is additional instruction at the A and B level, then the student contract is checked, and if he has elected a B or A, additional instruction is given, along with the A-B test. If he has not elected a B or A, a calculation is made to predict what he would probably get on the A-B test. If he is an underachiever who could get a B or A, he is provided with friendly advice and scored for approach if he takes it. He then has a chance to select among the "fun options" which might include such items as "More on that Topic" (which gives him A and B items not seen), a computational plotting program, a game film or a short film (videotapes can be shown on the TICCIT terminal). TIDBITS are historical vignettes, anecdotes, etc., related to the lesson topic, and are scored as approach. The student pays for fun options, using points earned through earlier achievement. The advisor may "advertise" certain options when the student arrives at the fun options menu. If he arrives at the fun options menu without having passed the mastery test, his choices are restricted and his bank account of points is low.

THE ADVISOR PROGRAM

The advisor is simply a set of routines which are either called by the student by depressing an interrupt key and typing a code word, or which are called by the course program at certain points. Four of the functions have been discussed above: strategy advice, status report, A-B advice, and "advertising." Other functions deal with setting and changing the grade contract, with scheduling reviews, and with keeping track of the "bank account" income and expenses.

MOTIVATIONAL TECHNIQUES

The reader will have observed a number of motivational techniques in the previous discussion. The whole concept of learner control, with emphasis on responsibility and skill in the use of strategies, has the potential of being very motivating to many students. The "So what?" file, the "fun options," and the point system used to earn fun options are designed to enhance motivation as well as to develop approach. In addition to these techniques, humor, cartoons, graphics, color, and audio are used throughout in an attempt to introduce the light and clever touches which have been used so well in the TV programs developed by Children's Television Workshop. Professional writers and TV scriptors are a part of the courseware teams. The intimate mixture of short videotapes and CAI sequences is a powerful new mechanism of expression for creative authoring talent, and we hope to learn to use it to the fullest to provide instruction that is palatable and fun as well as effective and efficient.

ON FREEDOM AND PRIVACY

The philosophy of learner choice and the respect for the individual implicit in our courseware design assure individual freedom. An individual is free even to remain antagonistic toward mathematics, or to resist attempts to accept responsibility, for our interactions are all persuasive rather than coercive.

On the issue of privacy, we plan to keep scores on mastery, speed and accuracy, learning strategies, voluntary approach, and the responsible use of control options. These scores are necessary both for the advisor program and for the human advisors to help each student achieve his objectives, and to improve the system for later students. The data are analogous to a combination of a grade transcript and a set of aptitude scores now found in students' files in registrars' offices at any college, but these data are more precise and detailed. They could presumably be misused in the same manner as could the registrars' files if security were lost. IQ scores have been misused and restrictions set up against their distribution. Presumably, appropriate restrictions must also be established to protect the misuse of our courseware scores. Indeed, it is possible to permit the individual to contract for the courseware to attempt to influence strategies, attitude, and responsibility as well as mastery. Such an agreement between the student and the system would be a positive and constructive substitute for the "privacy warning" now given to Dartmouth students whenever a record is kept of certain of their transactions at a terminal.

SUMMARY

Because of increasing cost problems in higher education, and the need to find more cost-effective approaches to traditional instruction, mainline instructional systems were proposed as a desirable alternative. The first reason discussed was economic. Mainline instructional systems, as defined in this paper, replace important subsystems within an educational institution with a less labor-intensive, potentially more cost-effective technology. Any proposal which uses technology as an adjunct to a classroom teacher, representing an add-on cost, is unlikely to represent a viable alternative in the years ahead unless somehow incorporated within a more radical restructuring of education.

That systematically designed mainline courseware can lead to the qualitative restructuring of curricula, and, in particular, can enhance independent, self-motivated learning and other important values was discussed. An example was taken from the TICCIT courseware development project in freshman mathematics to illustrate this discussion.

REFERENCES

1 V C ABBOUD C V BUNDERSON
A computer-assisted instruction program in the Arabic writing system
Technical Report No 4 Computer-Assisted Instruction Laboratory The University of Texas at Austin 1971

2 C V BUNDERSON
Justifying CAI in mainline instruction
In G P Weeg (Chairman) Conference on computers in the undergraduate curricula. Proceedings The University of Iowa Conference June 1970 Iowa City Iowa Center for Conferences and Institutes September 1970

3 C V BUNDERSON
The computer as a means toward controversial educational ends
In T E Kurtz (Chairman) Conference on computers in the undergraduate curricula. Proceedings Dartmouth Conference June 1971 Hanover New Hampshire University Press of New England 1971

4 P F DRUCKER
The age of discontinuity
New York Harper & Row 1969

5 W A JUDD C V BUNDERSON W BESSENT
An investigation of the effects of learner control in computer-assisted instruction prerequisite mathematics (MATHS)
Technical Report No 5 Computer-Assisted Instruction Laboratory The University of Texas at Austin 1970

6 T E KURTZ (Chairman)
Conference on computers in the undergraduate curricula
Proceedings Dartmouth Conference June 1971 Hanover New Hampshire University Press of New England 1971

7 A W LUEHRMANN
Dartmouth project coexist: the computer qua computer
In T E Kurtz (Chairman) Conference on computers in the undergraduate curricula Proceedings Dartmouth Conference June 1971 Hanover New Hampshire University Press of New England 1971

8 J R PIERCE (Panel chairman)
Computers in higher education: Report of the President's Science Advisory Committee
Washington DC US Government Printing Office 1967

9 A SMITH C GREGORY
MATHS user's manual
Computer-Assisted Instruction Laboratory The University of Texas at Austin 1970

10 S G TICKTON
The outlook for higher education in the big cities
Academy for Educational Development Inc 437 Madison Avenue New York New York 1969

11 G P WEEG (Chairman)
Conference on computers in the undergraduate curricula
Proceedings Iowa Conference University of Iowa Iowa City 1970

Should the computer teach the student, or vice versa?

by ARTHUR W. LUEHRMANN

Dartmouth College
Hanover, New Hampshire

This sermon begins with a parable.

Once upon a time in the ancient past there was a nation in which writing and reading had not yet been invented. Society was as advanced as possible, considering that it had no mechanism for recording the letter of the law or of writing agreements, contracts, or debts. Nor was there a way of recording the heritage of information and knowledge that had to be passed on from generation to generation.

As a result, a great fraction of the total effort of the society was spent in oral transmission of information. Master teachers, who themselves had been taught by older master teachers, lectured before children and young people of the society. Training a master teacher was a long and expensive process, and so the society could not afford many. For reasons of economy the curriculum was quite rigid and lectures were on a fixed schedule. Teaching, obviously, was a labor-intensive industry based on skilled, expensive talent. Education, per force, was a luxury that could be afforded by the elite classes only.

Then, one day, writing and reading were invented. Not surprisingly, the first application of this new technology was to business and government. Money was printed; laws were encoded; treaties were signed. In response to these needs, a reading and writing industry grew up. Within a few years it was able to offer a broad range of reading and writing services to its customers. The customers found this to be a convenient arrangement, since hiring readers and writers from service vendors eliminated the need for each customer to invest in an expensive R&D effort of its own. The customers remained illiterate.

At first the situation was somewhat chaotic. Each vendor of reading and writing service tended to develop its own favorite language and its own technique for encoding information, leading to incompatibilities that impeded the spread of the new technology. After a winnowing-out period, however, the number of competing systems settled down to a few and major difficulties were handled by translators—though inevitably something seemed to be lost in the process.

Always looking for new markets, the vendors of reading and writing service began to examine the area of education. In view of its elitist role in the society it had been dismissed at first as too limited a market. A few, more imaginative people, however, argued that the application of reading and writing technology could turn education into a mass market. They proposed the following plan of attack. Reading and writing specialists and master teachers would work as a team. The master teachers would deliver their best, most carefully prepared lectures to the reading and writing experts, who would write them carefully *verbatim* into books. The books would then be copied many times, and each copy would be made available to a new type of educational functionary—the *reader*. His only job would be to assemble groups of students and to read aloud to them the recorded lectures of the master teachers. In view of the fact that training such a reader would be far less expensive than the education of a master teacher, the on-going cost of such a program would be far less than that of the conventional lecture method. The new method came to be called Writing Assisted Instruction, frequently abbreviated to WAI.

Needless to say, WAI had its opponents. Established master teachers expressed doubt whether a less skilled reader would be able to communicate subtleties of inflection, and they were certain that a mere reader could not process student responses with skill or intelligence. WAI proponents counter-charged that the master teachers were merely expressing their vested interest in the present educational establishment, and, indeed, that they ought to be fearful because the superiority of WAI would ultimately drive out the conventional practitioners. Even within the education establishment some younger members became WAI supporters on the grounds that the new method was a boon to education research. Until then, teaching had been something of a black art, shrouded in the privacy of the classroom. To compare one teacher with another was impossible. But in the future, they said, the written record of the lectures of master teachers would make the teaching experience explicit and subject to analysis, comparison and improvement. It was high time, the young Turks exclaimed, that the teaching profession act with accountability to the public it served.

Unfortunately, such controversy remained for many years on a hypothetical plane. The number of actual WAI efforts was very small and their results were not striking. There was also a credibility problem. Many of the most outspoken advocates of WAI, especially in the legislature and in business and on local school boards, were themselves almost totally illiterate in the new reading and writing skills. How could they evaluate a new technology if they had not mastered it themselves?

Finally, government, business and some members of the education establishment decided to mount two or three large-scale demonstrations of WAI in order to show publicly the advantages of the new educational technology. For a period of several years curriculum experts collected information on a few key courses of lectures by assorted master teachers. The reading and writing experts wrote down the best series and read them aloud to the curriculum experts, who would criticize them and make improvements. The reading and writing experts would then incorporate the improvements in the next draft. Then came the field test. Readers began to read the drafts aloud to actual classes of students, and this led to further revision by the curriculum experts and rewriting by the reading and writing experts. At the end of a few more years a summative evaluation of the projects was undertaken by an independent, reputable educational testing organization, whose mission was to compare the cost and effectiveness of WAI with conventional education.

The parable is nearing its conclusion now. Actually it has two alternate endings, one happy and one sad. The sad ending, which follows now, is brief.

The educational testing organization reported that the projects were a complete vindication of Writing Assisted Instruction. It found that students taught by WAI performed even better on standardized tests than students taught by the average master teacher, that the students liked WAI better, and that the total cost of WAI was about a fourth that of conventional instruction. These pilot projects were imitated on a grand scale and education was revolutionized. Special institutes turned out vast numbers of readers and within ten years they were reading courses of lectures aloud to masses of people who could never have been educated before the new instructional technology arrived. The nation grew and prospered and thanked the day that the reading and writing industry was founded.

That is the sad ending. The happy ending is somewhat longer and more complicated. Here it is:

The educational testing organization found that WAI was neither measurably worse than conventional instruction, nor better. It found that costs were somewhat higher than anticipated, mainly because the market demand for people with reading and writing skills had driven their wages up near those of master teachers.

But this lukewarm finding was anticlimactic when it came, for the impact of reading and writing on education had taken a new turn during the intervening years. Here is how it happened.

At first a few master teachers had themselves found it necessary in pursuing their own research to spend the enormous effort required to master the skills of reading and writing. As they became more and more competent readers and writers, they began to see clearly the power of the written word within their own disciplines. Naturally enough the humanists were the first to apply this new intellectual tool to their fields of interest. Literature specialists collected stories, wrote them down, exchanged them with each other and began to develop literary criticism to a new height. Language specialists compiled lists of grammatical rules, which became writing manuals. Scientists were slower in becoming literate, with mathematicians leading the way, since they grasped the possibility of writing mathematical concepts in abstract notation. Nevertheless, for many years scientists continued to remain in verbal darkness.

While reading and writing had its primary impact on scholarly research, at the same time many master teachers across the land began to wonder whether it might not be beneficial to introduce elementary uses of reading and writing to students in their courses. A few language teachers began to show students how to write phrases and sentences, and the more venturesome teachers even asked students to write sentences of their own. Such experience, they claimed, greatly enhanced a student's understanding of syntax and rules of grammar. Even in subject areas far removed from language, to which reading and writing have a natural affinity, teachers began to report pedagogical gains due to having students carry out elementary reading and writing tasks as an adjunct to conventional instruction.

One obstacle to student use of reading and writing was the awkwardness of the main systems of notation, which had been developed mainly for research and business applications. The most popular such system was particularly difficult to format, since its characters all had to be positioned accurately in a fixed number of columns. Occasionally there were rumors that a group of teachers in a remote province near the northern frontier had developed a simpler writing system and all their students were using it daily. Such rumors were hard to verify; only a few people ever voyaged that far north, and, in any case, experts in the reading and writing industry seemed confident that anything that made the current system simpler would also take away its power and elegance. So most teachers adhered to it.

Within a few years teachers began to hold national meetings to tell one another how their students used reading and writing within their courses. Advocates of this type of use, which came to be called *adjunctive*, insisted that it be distinguished clearly from WAI. Writing Assisted Instruction, they charged, was nothing more than an improvement in the technology of delivering instruction. Adjunctive use of reading and writing by the student, on the other hand, represented a change in the intellectual content of instruction. They argued from the following philosophical premise:

> Reading and writing constitute a new and fundamental intellectual resource. To use that resource as a mere delivery system for instruction, but not to give a student instruction in how he might use the resource himself, was the chief failure of the WAI effort, they said. What a loss of opportunity, they exclaimed, if the skill of reading and writing were to be harnessed for the purpose of turning out masses of students who were unable to read and write!

WAI advocates responded that it was well and good that a few elitist schools teach their students the difficult skill of reading and writing; it was enough that WAI teach lesser skills to masses that might otherwise remain uneducated and unemployable.

How much longer, asked the WAI opponents in rebuttal, will an illiterate person be considered educated? How long will he be employable and for what jobs if elitist schools are turning out competent readers and writers by the hundreds?

The more visionary advocates of mass literacy told of foreseeing the day when students would spend more hours of the day reading and writing than listening to lectures. Small research libraries had indeed sprung up at some schools, but they were expensive operations limited to a few specialists who had to raise funds to pay for their use. Such people were particularly incredulous at the suggestion that every school ought to adopt as an educational goal the establishment of a significant library open freely to all students. School administrators were at first appalled at the idea that the library should not be on a pay-as-you-go basis but should be budgeted as part of the general institutional overhead costs.

But as time went on and even school administrators became competent and imaginative users of the skill of reading and writing, all schools gradually accepted as a mission the bringing of literacy to all students. Accreditation agencies examined the quality of libraries before approving schools. Books began to appear all over and finally even in people's homes. WAI did not die out altogether, but continued as a cost-effective alternative to the lecture. But as books reduced dependence on lectures, students made less use of both WAI and lectures and spent more time on their own reading and writing projects. The nation grew and prospered and wrote poems in praise of the day that reading and writing were discovered and made available to all people.

End of parable.

It is a perilous strategy, bordering on bad taste, to tell a joke and then for several pages explain why it was supposed to be funny. However, this allegorical tale has been told here not merely for entertainment but mainly for the moral lesson it carries. To compare reading and writing with computing might be dismissed as an amusing frivolity; but that would be wrong. Our fundamental philosophical premise here is that, like reading and writing,

> "[computing] constitutes a new and fundamental intellectual resource. To use that resource as a mere delivery system for instruction, but not to give a student instruction in how he might use the resource himself, has been the chief failure of the [C]AI effort. What a loss of opportunity if the skill of [computing] were to be harnessed for the purpose of turning out masses of students who were unable to [use computing]!"

As this example shows, it is a trivial editing task to go through the entire reading and writing fable and turn it into a story about computing and its uses in education. In fairness, the author admits that the story really *is* about computing and that reverse editing was done in the original telling so that it would seem to be about reading and writing. Yet, as a story about reading and writing it has considerable plausibility, doesn't it? The Writing Assisted Instruction program outlined in the story is not a totally absurd idea for putting reading and writing to use in education. One cannot argue against claims that committing lectures to writing would make education available to more people, would invite critical comparisons and a consequent improvement in subsequent revisions of written materials, and would be an asset to the study of the learning process itself. What does appear absurd, however, is the failure of these mythical WAI proponents to recognize that the best educational use of reading and writing is the teaching of reading and writing itself to everyone. Mass literacy is an educational mission about which few of us have doubts today.

Yet that consensus among us seems to vanish when one substitutes "computing" for "reading and writing" and "CAI" for "WAI". Mass computing literacy is not an agreed-upon educational goal. Today very few courses at any educational level show students how to use computing as an intellectual tool with applications to the subject matter being taught. Oh, there are a few isolated, subject-matter-free courses in computer programming; but their market is largely restricted to vocational-education students, at one end of the spectrum, and future computer professionals at the other. It is true that most schools consider it prestigious to have a large and powerful computer facility; but the fact of the matter is that such computers are usually the captives of research and administrative interests and operate on a pay-as-you-go basis. Ironically, it is in the most prestigious universities that students are least likely to be permitted to use those prestigious computers. It is a rare secondary school, college, or university that budgets and operates its computer facility in the same way that it budgets and operates its library. (There is a persistent rumor of an exceptional example in some remote province near the northern frontier, but so few people ever travel that way that the report is hard to verify.) In the main, literacy in computing simply is not an educational goal at many schools. Most educators seem to find bizarre the suggestion that accreditation agencies examine schools for the quality of their educational computing facilities, just as they now do with libraries.

The distressing truth today is that educators, local school boards and federal policy-makers are far more receptive to the plans of CAI proponents for using the technology of computing as a cost-effective delivery system for instruction in math or remedial English than they are to making computing itself a part of education. This statement should not be taken as a blast against CAI. On the contrary, CAI advocates are to be commended for their desire to reduce the cost of instruction, to tailor it to the different learning styles of students, to develop systems that encourage closer examination of what is being taught and systems for improving instruction, and to hold teachers and schools accountable to their clientele. With enough developmental work on CAI, it is likely that students will perceive the computer as a very superior teacher. Above all, CAI promises to make education a less labor-intensive industry and so to enable masses of people to become better educated. This is certainly a goal worth working for.

But there is a higher goal. If the computer is so powerful a resource that it can be programmed to simulate the instructional process, shouldn't we be teaching our students mastery of this powerful intellectual tool? Is it enough that a student be the subject of computer administered instruction—the end-user of a new technology? Or should his education also include learning to use the computer (1) to get information in the social sciences from a large data-base inquiry system, or (2) to simulate an ecological system, or (3) to solve problems by using algorithms, or (4) to acquire laboratory data and analyze it, or (5) to represent textual information for editing and analysis, or (6) to represent musical information for analysis, or (7) to create and process graphical information? These uses of computers in education cause students to become masters of computing, not merely its subjects.

It will be countered that such an educational mission is well and good for a few elitist schools, where students are willing to learn the difficult skill of computing; but it is enough that CAI teach lesser skills to masses of students that might otherwise remain uneducated and unemployable.

In response we ask, how much longer will a computer illiterate be considered educated? How long will he be employable and for what jobs if elitist schools are turning out competent computer users by the thousands?

The true story about computing and education is at its midpoint. Like the reading and writing parable, it has a sad ending and a happy ending. Which one actually occurs will be determined by you—teachers, school administrators, computer professionals, and government policy-makers.

Performance evaluation—A structured approach*

by STEPHEN R. KIMBLETON

University of Michigan
Ann Arbor, Michigan

INTRODUCTION

Computer systems are expensive in terms of development, acquisition and maintenance. The efficient and effective system appears to be the exception rather than the rule. Concern for system improvement is voiced by almost every computer-related management or technical support group. The types and amounts of data which can be gathered through hardware and software monitors appear to be ever increasing. A bewildering array of papers and proceedings are published every year and the problems of analyzing computer systems are described with loving detail.

In spite of the effort implicit in the activities described above, the author has rarely visited an installation at which the systems analysts were reasonably familiar with literature available in the technical journals on modeling computer systems. The most common approach to system improvement appears to be one of incremental enhancement coupled with a 'cut and try' approach. The absence of a basic underlying approach for a problem of such significance is difficult to believe.

MODELING SYSTEMS

In order to study large-scale complex systems efficiently, one normally proceeds via the development of suitable models. Effective tools for modeling have been developed by generations of operations research analysts. Unfortunately, mere knowledge of modeling tools does not guarantee the production of a good model of a system. Something akin to an O.R. viewpoint is also required. This viewpoint is normally opposed to that which seems reasonable to programmers and computing center managers by virtue of their training and natural inclinations. Consequently, a communication barrier is created between the O.R. analyst and those engaged in making decisions regarding the selection, evaluation and enhancement of computer systems.

This communication gap has two primary sources, the first being the difference in viewpoint. Most managers of computing centers have a reasonably thorough knowledge of the problems inherent in writing programs for currently existing large-scale computer systems. In particular, they are well aware that a meticulous concern for detail is necessary for the successful production of programs within acceptable time limits. Consequently, the forest of computing is viewed as a collection of individually interesting trees. This viewpoint produces a natural propensity toward simulation models of computer systems whose basic level of detail is in the sub-millisecond range.

A careful analysis of well-regarded models for other large-scale systems[1] demonstrates that the major prerequisite for successful modeling of complex systems is the identification of a reasonably small number of key variables. A corollary of this identification process is the omission of as many other variables as possible. Once the identification of the key variables driving the system has been performed, the basic methodologies of operations research, e.g., mathematical programming, queueing analysis, scheduling theory, etc., can be invoked. Clearly, the resulting model will appear incomplete to the casual observer knowledgable in the process being modeled. However, the acid test of the model is neither its complexity nor its apparent completeness; rather, it is the capability of the model for providing information of use by a decision-maker.

The resolution of the difference in viewpoints between the computing center manager and the O.R. analyst can only be achieved through construction by the latter of models which provide information of use to the former. This requires that the model relate the input variables to the performance of the system as a whole.

* This research was partially supported by the Office of Naval Research, Information Systems Program under Contract N000-14-67-0181-A-0036 (NR 049-311).

Unfortunately, most of the models existing in the literature are for components of a computer system. In particular, a large number of models exist for: (i) CPU scheduling algorithms,[2,3,4] (ii) memory management models[5,7,6] (iii) I/O models[8,9,10] and (iv) miscellaneous models of use in studying storage contention[11] the effects of buffering,[12] etc. A few attempts at building models which provide information on throughput have been made. In particular, the papers by Gaver[13] and Hanssmann, Kistler and Schultz[14] are noteworthy.

Before attempting to develop a model, it is natural to seek guidelines for the properties which it should possess if it is intended to be of use to a computing center manager. The tradeoff between the effort involved in building the model and the information obtained from it must be carefully considered. Consequently, an absolute formulation of guidelines cannot be made. However, extensive studies have been made of the categories of information required by a decision-maker in order to apply the formal techniques of decision analysis. In particular one must have:[15]

(1) a list of variables to be considered is making a decision, viz., the input variables and the output variables,
(2) a model showing the interaction between the input variables and the output variables,
(3) a list of constraints which the chosen solution must satisfy,
(4) a means for evaluating various alternative combinations of input and output variables.

As an example, the performance of a computer system is often measured in terms of:[16] (a) throughput, (b) turnaround time and (c) availability. Throughput is normally measured in jobs processed per week or month, availability is measured in terms of idle time per shift, and turnaround time is measured in terms of elapsed time from submission of a job until its output is available, Thus, it might be determined that a given system should possess the capability for processing 4000 jobs/month, with an average turnaround time of one hour and an availability of one hour/shift. In considering systems which meet this requirement, it is also desirable that some attention be given to the stability of various solutions. That is, given two systems which meet the requirements, one would normally choose the less expensive of the two. However, if a large increase in potential performance can be obtained at only a modest increase in cost, the more expensive system could be chosen.

Clearly, a model which effectively incorporates all of these factors constitutes a difficult objective. However, as a first cut at building a model of interest to a computing center manager, one might seek to relate the attributes of the programs which are to be run on the system (measured in some suitable manner) to the performance of the system as a whole (measured in terms of the three primary variables). In the following such an approach is described for a somewhat restricted class of programs.

THE SYSTEM PROCESS MODEL

A model for describing the performance of a computer system (hardware plus operating system) must possess the following two capabilities: (i) the activity of a process must be characterized in terms of some basic collection of descriptors, and (ii) the interaction of the collection of processes in the system must be characterized.

In modeling the activity of a process in a computer system, we observe with Denning[17] that at any given instant in time a job or process is in one of three states:

(i) active, i.e., currently being serviced by the CPU,
(ii) blocked, i.e., halted pending completion of some event and incapable of being serviced by the CPU,
(iii) ready, i.e., awaiting the CPU.

The lifetime of a job consists of an alternating sequence of active, blocked and ready intervals, which may have length zero. The utility of this observation does not appear to have been previously noted. Perhaps, this is because of observed variations in the lengths of these intervals, the relative times at which they occur and their number. That is, the activity of a process is not deterministically reproducible when its evolution is viewed as an alternating sequence of such intervals. Fortunately, the activity of the process is statistically reproducible.

The interactions of the collection of concurrently executing processes are assumed to behave in the following manner:

(i) all jobs are statistically identical,
(ii) all processes cycle through the active, blocked and ready states in the same order, i.e., processes do not pass each other, and an exit from the active state results in an entrance into the blocked state,
(iii) the amount of time spent in any given state is exponentially distributed.

Although these assumptions are a simplification of the manner in which a computer system operates, some available evidence[19] seems to indicate that estimates for the performance of a real system which can be obtained through the use of this model are reasonable. In the following, we shall refer to this model of the activity of a computer system as the System Process Model (SPM).

The preceding assumptions are made in order to obtain an analytically tractable model. Results similar to those described later in the paper can be obtained for non-statistically identical processes. However, statement of the results becomes significantly more complex. The assumption that processes do not pass each other is clearly not satisfied by the operation of a real system. However, the effects of passing appear to be local in nature and, consequently, this assumption does not appear to be unduly restrictive. The exponential assumptions for the time spent in the active, blocked and ready states are only tenuously satisfied by some available data.[19] Thus, this assumption does not appear to be unduly restrictive on the range of applicability of this approach.

For the SPM, mathematical expressions for the CPU utilization, the average number of active intervals, the average length of a ready interval and the average system residence time of a process can be derived. In obtaining these results, the average length of an active interval, the average length of a blocked interval and the total amount of CPU time are assumed to be known. The results obtained can then be used as estimators for the corresponding quantities of a real system. In order to clearly distinguish between quantities which are obtained for the SPM and those which are assumed to be obtained through the use of monitors from a real system, we adopt the following notation:

(a) $\hat{U}(K)$ denotes the SPM estimate for $U(K)$, the CPU utilization given K concurrently executing jobs in the system,

(b) $\hat{N}$ denotes the SPM predictor for $\bar{N}$, the average number of active intervals,

(c) $\hat{R}(K)$ denotes the SPM estimate for $\bar{R}$, the average ready interval,

(d) $\hat{S}(K)$ denotes the SPM estimate for $\bar{S}$, the average system residence time of a job, i.e., the elapsed time from initiation of the first active interval until termination of the last active interval.

The following theorem has been obtained elsewhere.[18,19]

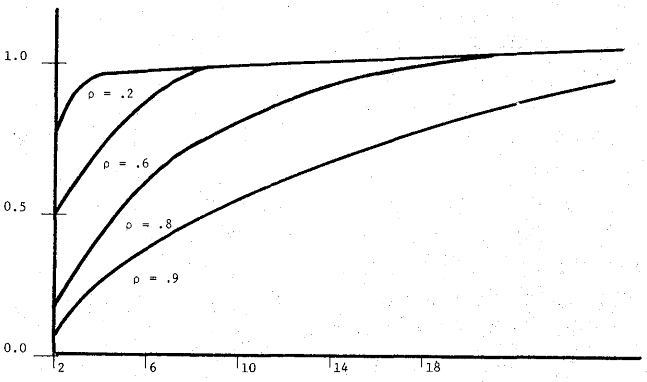


Figure 1—CPO utilization estimate

Theorem:

Let A_Σ denote the total amount of CPU time required by a given process. Let $\bar{A}$ and $\bar{B}$ denote the average length of the active and blocked intervals of the process. For the System Process Model with K concurrently execution processes:

(i) $\hat{N} = A_\Sigma/\bar{A}$,

(ii) $\hat{U}(K) \geq 1 - \rho^{K-1}$

(iii) $\hat{R}(K) = (K - E[I(K)])\, \bar{A}\, \hat{U}(K)$,

(iv) $\hat{S}(K) = \hat{N}(\bar{A} + \bar{B} + \hat{R}(K))$.

where $\rho = \bar{B}/(\bar{A}+\bar{B})$ and $I(K)$ denotes the number of active intervals which can be initiated during a blocked interval.

The behavior of $\hat{U}(K)$ is described in Figure 1. We note that the effect of increasing the number of concurrently executing processes in the system is to increase $\hat{R}(K)$ in a nearly linear manner, provided the CPU utilization is near one. We further observe that since A_Σ represents the total amount of CPU time required by a job, variations in A_Σ with different injections of the same job will reflect the effects of system overhead and should be nearly negligible.

Throughput is a subject of much interest at most computer installations. Unfortunately, since it is normally computed in terms of jobs per day, week or month, it is difficult to examine the impact of a single job on the throughput at an installation. However, throughput is simply the rate at which jobs are processed. An expression for this quantity can be obtained through the use of our model. Indeed, by using well-known arguments from renewal theory,[20] it follows that the rate at which one job is completed is $1/\hat{S}$. Since there are K distinct jobs in the system, the rate at which these jobs are completed is $K/\bar{S}$. Thus, under the assumptions of our model, the throughput is $K/\bar{S}$. It should be observed that this expression is consistent with the definition of throughput. Over a time interval of length S, precisely $K/\bar{S} \times \bar{S} = K$ jobs will have been completed.

It is appropriate to inquire as to the stability of decisions achieved through reasoning in the preceding manner. Fortunately, the available evidence seems to indicate that process behavior is statistically reproducible, provided the system is large enough so that no single job dominates it. Indeed, measurements made on the University of Michigan's MTS Computer System (a twin-processor 360/67 with three 2301 paging drums, three eight-drive 2314 disks, two data cells, 100 terminals, plus assorted other peripheral devices) through the use of the embedded software monitor[21] have shown that the probability distributions obtained for the lengths of the active, blocked and ready intervals vary in a very predictable manner as a function of the CPU load. (The MTS System is a processor bound system; hence it is reasonable that the CPU load should be the controlling influence on the variation in the distributions of the lengths of these intervals.) Similar arguments would appear to hold for other systems in which there exists a well-defined limiting resource other than the CPU.[19] If there is no limiting resource, the system is lightly loaded and the estimates described here may not be valid since processes can 'chase' each other.

EXTENSIONS AND APPLICATIONS

The preceding results are of use in studying the effects of different job mixes for a computer system in which the hardware and operating system are assumed fixed. If one wishes to study the effects of varying either the number, type or arrangement of I/O devices, it is useful to have a means of estimating $\bar{B}$. In addition, if one wishes to study the effect of changes in the CPU and/or operating system, it would also be desirable to have a method for estimating $\bar{A}$. Some considerations relevant to obtaining such estimators will now be described.

Estimation of $\bar{A}$

Both the 'power' of the CPU and the impact of the operating system are reflected in $\bar{A}$. Consequently, of the four basic approaches to gathering information about a computer system, i.e., (a) benchmarks, (b) hardware and software monitors, (c) simulation models and (d) analytic models, the use of analytic models would seem least likely to succeed. This stems from the difficulty of treating dependent relationships with analytic models. Attempting to predict $\bar{A}$ as a function of the hardware and operating system characteristics through the use of a simulation model could probably be done. However, a rather fine level of detail might have to be incorporated in order to achieve a reasonable estimator. Since one is normally interested in making a large number of comparisons, this might render the cost of a simulation approach prohibitive. It should be noted, however, that it would be of independent interest to study the minimal amount of detail which must be incorporated in a simulation model in order to yield consistent estimators for $\bar{A}$. Such a study might provide some information on the level of detail which should be incorporated in a simulation model of a computer system—a subject on which little appears to be known.

Perhaps the most suitable approach to this problem would be through the use of hardware or software monitors applied to suitably chosen benchmarks. Unfortunately, the difficulties encountered in identifying the process responsible for a particular event being measured with a hardware monitor appears to make their use somewhat difficult. A software monitor capable of tracing the activities of an individual process such as the event driven software monitor embedded in the MTS System[21] would appear to be ideal. However, most of the commercially available software monitors operate on a sampling basis which necessitates a certain amount of guesstimating in determining $\bar{A}$, although these monitors are useful for determining fractional CPU utilization.

Estimation of $\bar{B}$

Estimating $\bar{B}$ appears to be rather simpler than estimating $\bar{A}$. Data described elsewhere[19] indicate that variations in the length of $\bar{B}$ are rather small (at least for a system in which the CPU is the limiting resource) over a fairly large CPU load range. A simulation[22] of a single channel eight drive IBM 2314 indicates that the distribution of arriving requests by module is the most significant single factor in achieving a satisfactory channel utilization. In particular, given a suitable distribution of requests by module, the delay time of a request appears to be rather insensitive to the particular disk scheduling policy being used, e.g., FCFS (first come first served), SCAN,[10] or SSTF.[23]

This result would seem to indicate that in estimating $\bar{B}$ as a function of the collection of I/O devices in the system, the level of detail which must be incorporated is not excessive. In particular, in addition to knowledge of the hardware characteristics of the devices, the most important variables appear to be: (a) arrival rate by device, (b) distribution of arrival requests by cylinder for disks, (c) device-channel arrangement and (d) degree of buffering. Given this information one may write:

$$\bar{B} = \sum p_i \bar{B}_i.$$

In this expression p_i is the fraction of blocked intervals which occur for a device of type i, $\bar{B}_i$ denotes the average length of a blocked interval for such a device and the sum is computed over all I/O devices in the system. Estimation of $\bar{B}_i$ is straightforward for dedicated peripherals if the distribution of arrivals to the device and the amount of information to be transmitted in each request is known. For drums and disks, comments made earlier would seem to indicate that rather simple simulation models can be employed. Alternatively, somewhat less accurate estimates could be obtained through the use of analytic models described in the literature.

Block times less than those indicated through the use of these models are indicative of either a good arrangement of data sets on devices or an effective use of buffers. Indeed, the appropriate depth of buffering might be examined by observing the extent to which modifications in this depth affect the observed blocked interval. An observed value of the average blocked interval greater than that indicated by the device models may indicate poor data set organization for a given job, contention between jobs for data sets stored on the same device, channel contention, etc. Although this approach does not pinpoint the exact source of delay, it can be used as an indicator for the presence of potential problems. Further, comparison of the values observed with those predicted by the models provide insight into the size of the anticipated reward which might be achieved through further investigation.

SUMMARY

We have obtained a model which yields estimates for turnaround time, CPU availability and throughput rate for a collection of statistically identical processes characterized in terms of the total amount of CPU time required and the average lengths of the active and blocked intervals. Consequently, the effects of varying the number of users in the system upon the three primary measures of system performance can be investigated. In turn, this allows one to apply the techniques of formal decision analysis to the selection, evaluation and comparison of computer systems. Such an application requires an expression of the means to be used in judging various alternatives and the decision to be made.

REFERENCES

1 E S QUADE
Analysis for military decisions
Technical Report R-387-PR The RAND Corporation Santa Monica California November 1964 AD 453 887

2 J M McKINNEY
A survey of analytical time-sharing models
Computing Surveys 1969 Vol 1 No 2 pp 105-116

3 L KLEINROCK
Analysis of a time-shared processor
Naval Research Logistics Quarterly 1964 Vol 11 No 10 pp 59-73

4 E G KOFFMAN
Analysis of two time-sharing algorithms designed for limited swapping
J ACM 1968 Vol 15 No 3 pp 341-353

5 P J DENNING
Virtual memory
Computing Surveys 1970 Vol 2 No 3 pp 153-189

6 R L MATTSON J GECSEI D R SLUTZ I W TRAIGER
Evaluation techniques for storage hierarchies
IBM Systems Journal 1970 Vol 9 No 2 pp 78-117

7 T PINKERTON
Program behavior and control in virtual storage computer systems
TR No 4 CONCOMP Project University of Michigan April 1968

8 E G KOFFMAN
Analysis of a drum input/output queue under scheduled operation in a paged computer system
J ACM 1969 Vol 16 No 1 pp 73-90

9 J ABATE H DUBNER S B WEINBERG
Queueing analysis of the IBM 2314 disk storage facility
J ACM 1968 Vol 15 No 4 pp 577-589

10 T J TEOREY T B PINKERTON
A comparison analysis of disk scheduling policies
Proceedings Third Symposium on Operating System Principles pp 114-121 1971

11 C E SKINNER J R ASHER
Effects of storage contention on system performance
IBM Systems Journal 1969 Vol 8 No 4 pp 319-337

12 W W CHU
Buffer behavior for poisson arrivals and multiple synchronous constant outputs
IEEE Transactions on Computers 1970 Vol C-19 No 6 pp 530-534

13 D GAVER
Probability models for multiprogramming computer systems
J ACM 1967 Vol 14 No 3 pp 423-438

14 F HANSSMANN W KISTLER H SCHULTZ
Modeling for computing center planning
IBM Systems Journal 1971 Vol 10 No 4 pp 305-324

15 D TEICHROEW
An introduction to management science: Deterministic models
J Wiley New York New York 1964

16 P CALINGAERT
System performance evaluation—Survey and appraisal
Comm ACM January 1967 Vol 10 No 1 pp 12-18

17 P DENNING
The working set model for program behavior
C ACM 1968 Vol 11 No 5 pp 323-333

18 S R KIMBLETON C G MOORE
A probabilistic approach to systems performance evaluation
Proceedings of the ACM SIGOPS Workshop on Systems Performance Evaluation 5-7 April 1971

19 S R KIMBLETON C G MOORE
A limited resource approach to system performance evaluation
Technical Report No. 71-2 ISDOS Research Project/Performance Modeling Group Department of Industrial Engineering The University of Michigan Ann Arbor Michigan 1971

20 S M ROSS
Applied probability models with optimization applications
Holden-Day 1970

21 T B PINKERTON
The MTS data collection facility
Memorandum 18 CONCOM Project University of Michigan June 1968

22 M MEHTA
Unpublished Memorandum Department of Industrial Engineering The University of Michigan Ann Arbor Michigan 1971

23 P J DENNING
Effects of scheduling on file memory operations
SJCC 1967 pp 9-21

Protection—Principles and practice*

by G. SCOTT GRAHAM and PETER J. DENNING

Princeton University
Princeton, New Jersey

INTRODUCTION

The protection mechanisms of computer systems control the access to objects, especially information objects. The range of responsibilities of these mechanisms includes at one extreme completely isolating executing programs from each other, and at the other extreme permitting complete cooperation and shared access among executing programs. Within this range one can identify at least seven levels at which protection mechanisms can be conceived as being required, each level being more difficult than its predecessor to implement:

1. No sharing at all (complete isolation).
2. Sharing copies of programs or data files.
3. Sharing originals of programs or data files.
4. Sharing programming systems or subsystems.
5. Permitting the cooperation of mutually suspicious subsystems—e.g., as with debugging or proprietary subsystems.
6. Providing "memoryless" subsystems—i.e., systems which, having performed their tasks, are guaranteed to have kept no secret record of the task performed (an income-tax computing service, for example, must be allowed to keep billing information on its use by customers but not to store information secretly on customers' incomes).
7. Providing "certified" subsystems—i.e., those whose correctness has been completely validated and is guaranteed *a priori*.

We shall consider here the protection mechanisms required for level 5, but not those required for levels 6 or 7. We do this because we are interested in specifying the structure of protection mechanisms that work irrespective of considerations of internal program structure; levels 6 and 7 in general require control over internal program structure. Moreover, little is known about the mechanisms for levels 6 and 7, whereas much is known about the mechanisms for levels 1 to 5.

Much has been written about the legal and social implications of protection and privacy (see, for example Reference 10). We deal exclusively with the technical aspects of protection—i.e., the procedures governing the access of executing programs (processes) to various resources in the system. This is because little has been written about the technical side of the problem. It is also because the technical approach, when balanced with the legal approach, is useful in clarifying the inadequacies of both approaches and in understanding how they can complement one another. For example, without a technically sound protection system, a user might find it possible to perform malicious acts undetected, rendering certain laws unenforceable. Even with a sound protection system, however, certain aspects of protection can be dealt with only by laws, e.g., the problem of false identification.

The first step in the design of a protection system is the specification of the main parties and their interactions. That is, the entities to be protected and the entities to be protected against are identified, and rules by which the latter may access the former are formulated. For this, common sense and examples of existing protection systems can be used to guide our thinking in the development of the elements and principles of a protection system. We formalize these principles as a model (theory) of protection, based on one developed by Lampson.*[16] Perhaps the most important aspect of

* Work reported herein was supported in part by NASA Grant NGR-31-001-170 and by National Research Council of Canada Grant A7146.

* Approximately half our development here is an exposition and reformulation of the elements of Lampson's model. The remainder extends Lampson's in several important ways. We have investigated the problems of establishing the correctness of a protection system, of creating and deleting objects, of implementing cooperation among mutually suspicious subsystems, and of identifying the existence or absence of the elements of the model in contemporary systems.

the model is the notion that each process has a unique identification number which is attached by the system to each access attempted by the process. As will be seen, this will make it impossible for a process to disguise its identity and will allow us to establish the correctness of the model with respect to implementing protection at level 5.

The implications of the protection model with respect to operating system structure and hardware architecture will be considered. Although the model is abstracted from examples of existing systems, we shall see that few current commercial systems meet the requirements of a complete (secure) protection system. We shall identify a few existing systems which do meet the completeness requirement of our model.

A THEORY OF PROTECTION

In current systems, protection takes many forms. The IBM System/360 uses a "key-lock" approach to memory protection.[12] A "ring" protection mechanism is used in Multics.[9,18] The Hitac 5020 Time-Sharing System uses a key-lock-ring mechanism.[17] There are many other examples. As Lampson has pointed out, the myriad of implementations forces us, as observers, to take an abstract approach to the subject of protection;[16] otherwise, we might never make sense out of the seemingly endless variety of solutions to protection problems.

Elements of the model

A protection system comprises three parts, which will be reflected as components of our model. The first component is a set of *objects*, an object being an entity to which access must be controlled. Examples of objects are pages of main memory, files, programs, auxiliary memory devices, and instructions. Associated with each object X is a unique *identification number* which is assinged to the object when it is created in the operating system. Identification numbers might, for example, be derived from a clock which measures time in microseconds since the system was first built; the time-of-day clock on the IBM System/370, being a 52-bit counter, could allow a new object name every microsecond for approximately 143 years.[13] We shall use the symbol X interchangeably for an object's name and number.

The second component of the protection model is a set of *subjects*, a subject being an active entity whose access to objects must be controlled. A subject may be regarded as a pair (process, domain), in which a process is a program in execution and a domain is the protection environment (context) in which the process is operating. Examples of domains include supervisor/problem-program states in IBM System/360,[12] the 15 user environments in IBM's OS/360-MVT,[11] or the file directories and rings in Multics.[1] Other terms which have been used to denote the idea of domain are "sphere of protection" [4] and "ring." [9,18] Since subjects must be protected, they too are objects, and each has a unique identification number.

The third component of a protection system is the rules which govern the accessing of objects by subjects. We shall describe a particular choice of rules shortly. These rules are the heart of the protection system. The rules must be simple, allowing users to gain an immediate understanding of their scope and use. They must be complete, not allowing a subject to gain unauthorized access to an object. They must be flexible, providing mechanisms which easily allow the desired degree of authorized sharing of objects among subjects.

An important property of our model is: each and every attempted access by a subject to an object is validated. This is necessary in order to permit cooperation among mutually suspicious subjects, for it cannot otherwise be guaranteed that a previously well-behaved subject which suddenly turned malicious or went awry will be denied access when authorization is lacking.

The choice of the subjects, objects, and rules of a protection system is at the discretion of the system designer and will be made to meet the protection and sharing requirements of the given system. We return later to the problems of choosing subjects and objects.

It is convenient to regard all the information specifying the types of access subjects have to objects as constituting a "protection state" of the system. There are three distinct problems to be solved: representing the protection state, causing subjects to access objects only as permitted by the protection state, and allowing

		OBJECTS						
		subjects			files		devices	
		S_1	S_2	S_3	F_1	F_2	D_1	D_2
SUBJECTS	S_1		block wakeup		read write		seek	
	S_2			stop		update		seek
	S_3				delete	execute		

Figure 1—Portion of an access matrix

subjects to alter the protection state in certain ways. With respect to the first, we may represent the protection state of the system as an *access matrix* A, with subjects identifying the rows and objects the columns (see Figure 1). The entry $A[S, X]$ contains strings, called *access attributes*, specifying the access privileges held by subject S to object X. If string α appears in $A[S, X]$, we say "S has α access to X." For example, in Figure 1, subject S_1 may read file F_1, since 'read' appears in $A[S_1, F_1]$; or, S_2 may stop S_3. Figure 2 shows that an alternative representation of the protection state is a directed graph, which is in one-to-one correspondence with the access matrix.

Associated with each type of object is a *monitor*, through which all access to objects of that type must pass to be validated. Examples of monitors are the file system for files, the hardware for instruction execution and memory addressing, and the protection system for subjects. An access proceeds as follows:

1. S initiates access to X in manner α.
2. The computer system supplies the triple (S, α, X) to the monitor of X.
3. The monitor of X interrogates the access matrix to determine if α is in $A[S, X]$; if it is, access is permitted, otherwise, it is denied and a protection violation occurs.

Note that access attributes are interpreted by object monitors at the times accesses are attempted. Figure 3 shows the organization of the protection system. The mechanisms between the dashed lines of that diagram are invisible to subjects—subjects direct their references to objects, these references being intercepted and validated by the monitors of the protection system.

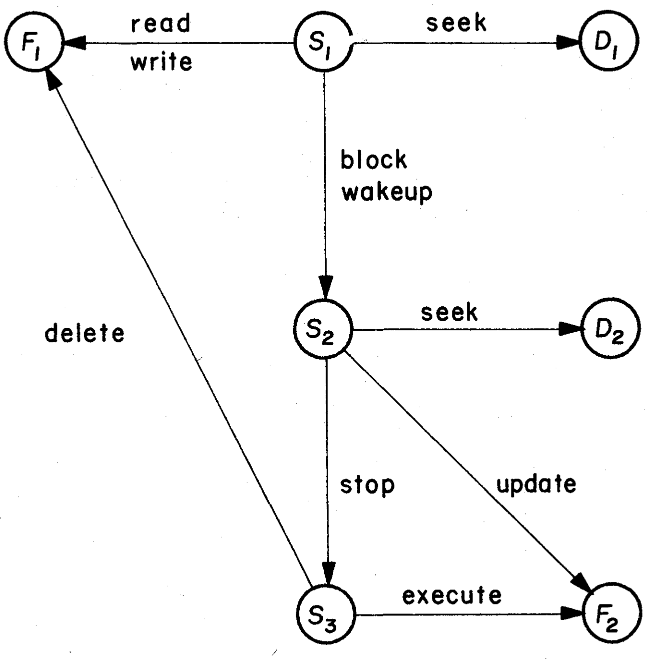


Figure 2—Access diagram

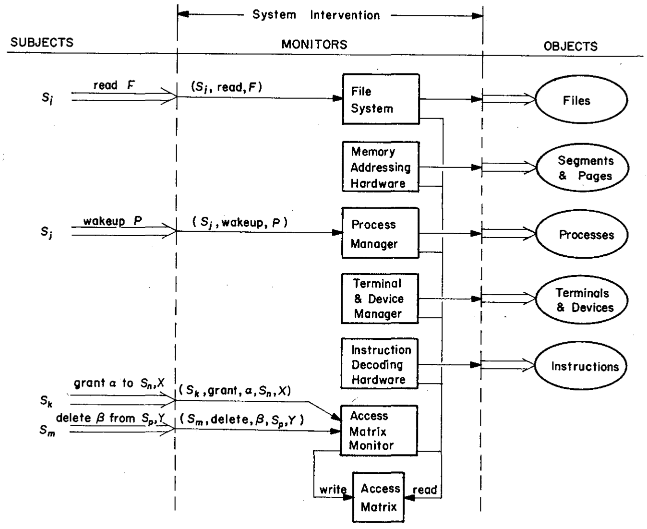


Figure 3—Organization of protection system

It is important to note that the identification number of a subject is system-provided in rule 2 above, and cannot be forged by S. That is, even if every subject knows the identification number of every other subject, there is no way for S to alter the fact that its identification number is presented to the monitor in rule 2; hence the monitor cannot be "tricked" into interrogating the wrong entry of A. Since no subject may access A, it is on this basis that we can develop proofs concerning the correctness of the protection system. The correctness of the protection system can be reduced to the correctness of the monitors.

The foregoing rules govern the use, by monitors, of the access matrix, once it has been specified. We require rules for changing the access matrix itself. These rules will be enforced by the monitor of the access matrix. Unlike the monitors of system objects, the access matrix monitor may modify the access matrix. In particular, it may transfer, grant, or delete access attributes on command of subjects and only on appropriate authorization. For this purpose, we introduce the access attributes 'owner' and 'control,' and the notion of a *copy flag* (denoted by asterisk), and the rules R1-R3 of Table I to be implemented by the access matrix monitor.

TABLE I—Protection System Commands

Rule	Command (by S_o)	Authorization	Operation
R1	**transfer** $\begin{Bmatrix}\alpha^*\\ \alpha\end{Bmatrix}$ **to** S, X	'α*' in $A[S_o, X]$	store $\begin{Bmatrix}\alpha^*\\ \alpha\end{Bmatrix}$ in $A[S, X]$
R2	**grant** $\begin{Bmatrix}\alpha^*\\ \alpha\end{Bmatrix}$ **to** S, X	'owner' in $A[S_o, X]$	store $\begin{Bmatrix}\alpha^*\\ \alpha\end{Bmatrix}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	delete α from $A[S, X]$
R4	ω:= **read** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	copy $A[S, X]$ into ω
R5	**create object** X	none	add column for X to A; store 'owner' in $A[S_o, X]$
R6	**destroy object** X	'owner' in $A[S_o, X]$	delete column for X from A
R7	**create subject** S	none	add row for S to A; execute **create object** S; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_o, S]$	delete row for S from A; execute **destroy object** S

Rule 1 permits a subject to transfer any access attribute it holds for an object to any other subject, provided the copy flag of the attribute is set, and it may specify whether the copy flag of the transferred attribute is to be set; in Figure 4, for example, S_1 may place 'read*' in $A[S_2, F_1]$ or 'write' in $A[S_3, F_1]$, but it may not transfer its ability to block S_2 to any other subject. The purpose of the copy flag is preventing an untrustworthy subject from wantonly giving away access to objects. Rule R2 permits a subject to grant to any subject access attributes for an object it owns; in Figure 4, for example, S_1 can grant any type of access for S_2 to any subject, or any type of access to device D_2 to any subject. Rule R3 permits a subject to delete any access attribute (or copy flag) from the column of an object it owns, or the row of a subject it controls; in Figure 4, for example, S_1 can delete any entry from columns S_2 or S_3, or from row S_3. In order to facilitate these rules, we require that 'control' be in $A[S, S]$ for every subject S and we include rule R4, which permits a subject to read that portion of the access matrix which it owns or controls. Rules R5-R8, which govern the creation and deletion of subjects and objects, will be discussed shortly.

It should be noted that a subject may hold 'owner' access to any object, but 'control' access only to subjects. For reasons to be discussed shortly, we shall assume each subject is owned or controlled by at most one other subject, though other multiply-owned non-subject objects are allowable. If a subject has 'owner' access to another subject, the former can grant itself 'control' access to the latter, so that 'control' is implied by 'owner.' It is undesirable for a subject to be 'owner' of itself, for then (by Rule R3) it can delete other subjects' access to itself.

The rules R1-R4 and access attributes shown in Figures 1 and 4 should be interpreted as examples of rules which can be provided for the purposes intended. One might, for example, introduce a "transfer-only"

	S_1	S_2	S_3	F_1	F_2	D_1	D_2
S_1	control	owner block wakeup	owner control	read* write*		seek	owner
S_2		control	stop	owner	update	owner	seek*
S_3			control	delete	owner execute		

Figure 4—Extended access matrix

mode by postulating a "transfer-only copy flag"; if this flag is denoted by the symbol •, then in R1 the command to transfer α (or α•) from S_0 to S for object X would be authorized if α• were in $A[S_0, X]$ and would cause α • to be deleted from $A[S_0, X]$ and α (or α•) be placed in $A[S, X]$. This is useful if one wants to limit the number of outstanding access attributes to a given object—e.g., if it is required that each subject be owned by at most one other subject, or that a given object be accessible to a limited number of subjects. One can also define "limited-use" access attributes, of the form (α, k); access in manner α is permitted only if $k>0$, and each use of this attribute decreases k by one.

Creation and deletion of subjects and objects

The creation of a non-subject object, e.g., a file, is straightforward, consisting of adding a new column to the access matrix. The creator subject executes a command (rule R5 of Table I) and is given 'owner' access to the newly created object; it then may grant access attributes to other subjects for the object according to rule R2. The destruction of an object, permitted only to its owner, corresponds to deleting the column from the access matrix (rule R6).

Creating a subject consists of creating a row and column for the new subject in the access matrix, giving the creator 'owner' access to the new subject, and giving the new subject 'control' access to itself (rule R7). The destruction of a subject, permitted only to its 'owner,' corresponds to deleting both the row and the column from the access matrix (rule R8).

According to our definition of subject as a pair (process, domain), there are two ways a subject can come into existence: a given process switches from one domain to another, or a new process is created in some domain. (Similarly, there are two ways a subject can go out of existence.) With respect to the former problem, a process must have authorization to switch domains since domain-switching entails a gain or loss of access attributes. Specifically, if process P wishes to switch from domain D_1 to domain D_2, subjects $S_1=(P, D_1)$ and $S_2=(P, D_2)$ would exist in the system; the switch by P from D_1 to D_2 is permitted only if authorization (e.g., access attribute 'switch') is in $A[S_1, S_2]$. In practice, it would be more efficient to regard a subject as being dormant (rather than nonexistent) when its process is operating in another domain—i.e., if P switches from D_1 to D_2, subject (P, D_1) becomes dormant and subject (P, D_2) becomes active, but the switch neither creates a new subject nor destroys an old one. (The idea is very much like that of transferring control between coroutines.) In practice, therefore, rules R7 and R8 would be invoked only when a process is created or destroyed.

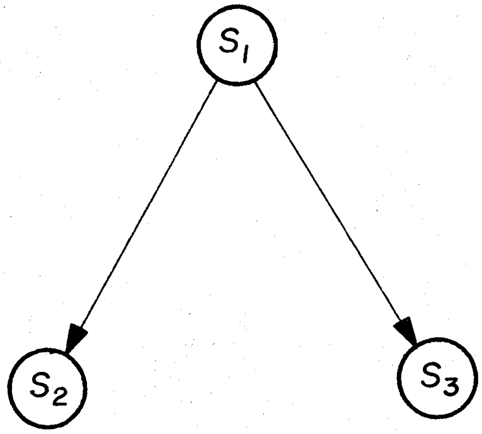


Figure 5—Ownership diagram

We suppose that the system enforces the requirement that each subject be owned by at most one other subject—i.e., an owner may not grant 'owner,' and 'owner' is either untransferable or is transfer-only. In this case the relation 'owner' defines naturally a tree hierarchy of subjects. Figure 5 shows the subject tree for the access matrix of Figure 4. If S_1 is 'owner' of S_2, S_2 is *subordinate* to S_1. By rule R1, a subject can transfer only a subset of its access attributes to a subordinate; by rule R2, it can grant any access for a subordinate to any other subject; by rule R3, it can delete any access attribute from a subordinate; and by rule R4, it can discover what access attributes a subordinate has. We have carefully avoided including any provision for a subject to acquire for itself (copy) an access attribute it does not have but which a subordinate has obtained. This can be incorporated, if desired, as an extension to the model.

It is not overly restrictive to require that subjects be members of a hierarchy, the utility of hierarchies having been demonstrated adequately in practice. This mechanism can be used to ensure that a subordinate, upon creation, has no more privileges than its creator. It can be used to define a protocol for reporting protection violations (and indeed any other fault condition), for the violation can be reported to the immediate superior of the subject generating it. It can be used as an aid in resource allocation and accounting, for a subordinate can be granted only a subset of the resources held by its creator. Considerations such as these motivated the design of the RC4000 system.[2] Our subject

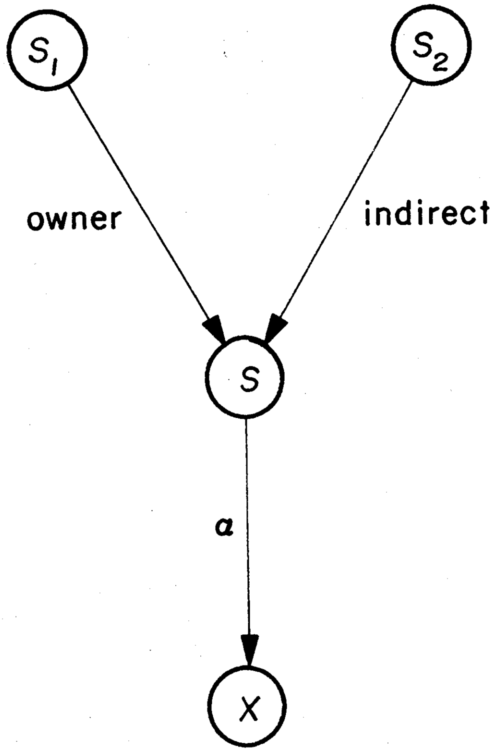


Figure 6—Sharing an untrustworthy subsystem

hierarchy is in fact the same as the sphere-of-protection hierarchy proposed by Dennis and Van Horn,[4] or the domain hierarchy used by Lampson.[15] If one regards a process operating in the context of a file directory as a subject, then the directory hierarchy of Multics defines another example of a subject hierarchy.[1] As will be seen, the rings of Multics also define a subject hierarchy.[9,18]

A subject in a hierarchy without an owner is called a *universal subject.* For convenience, we assume there is only one universal subject, and it has every possible access attribute to every object. When a user wishes to sign on the system, he communicates with the universal subject. Having convinced itself of the user's identity, the universal subject will create a subordinate subject for the user, and will initialize the row of the access matrix corresponding to that subject with attributes derived from some data file private to the universal subject.

Sharing untrustworthy subsystems

It is clear from the above that, if subject S_1 wishes to share an object X with subject S_2, it may do so without risk of S_2 acquiring access to any other of its objects. But if X is a subject, we must ensure that S_2 is allowed access to objects X can access and that X, if an active entity, cannot access objects of S_2 without authorization.

Suppose subject S_1 owns a subsystem S which it wishes to share with subject S_2. However, S_2 does not trust S, and S_1 may wish to revoke S_2's access to S at any time. We introduce the access attribute 'indirect' for use among subjects: if one subject has 'indirect' access to another, the former may access objects in the same manner as the latter, and it may read but not acquire for itself access attributes of the latter; the subject which owns the latter may (rule R3) revoke the former's 'indirect' access at any time. The sharing of S among S_1 and S_2 according to this idea is illustrated in Figure 6. The rules of operation of object monitors must be modified slightly to allow for indirect access:

1. S_2 initiates indirect access to X through S in manner α;
2. The system supplies (S_2, α, S-X) to the monitor of X;
3. The monitor of X interrogates the access matrix to determine if 'indirect' is in $A[S_2, S]$ and α is in $A[S, X]$; if so access is permitted, otherwise it is denied and a protection violation occurs.

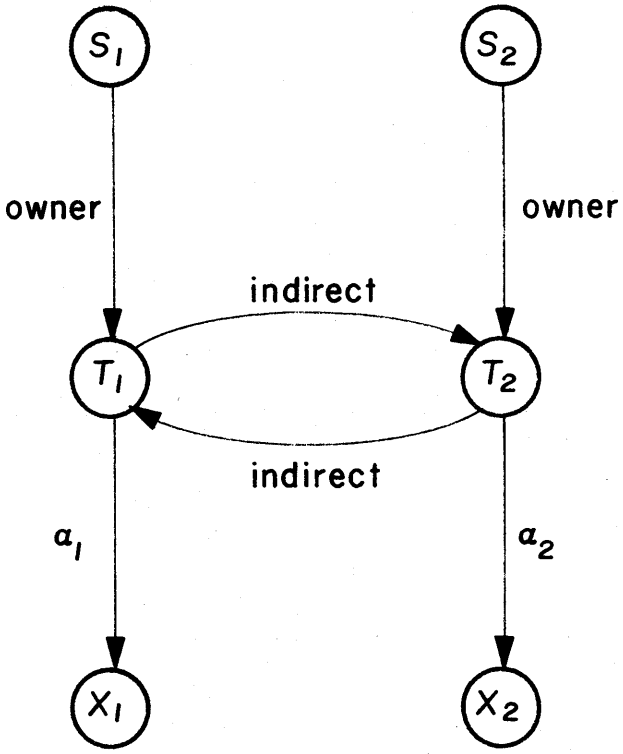


Figure 7—Cooperation between mutually suspicious subsystems

An example of the type of sharing in Figure 6 is the "debugging problem," in which S_1 wishes S_2 to debug S. Here S_2 needs complete access to all objects accessible to S, but S must be denied access to S_2 or its other objects. Another example of the same type of sharing is the "grading program" problem, where S_1 corresponds to a student who is submitting a program S to an instructor-program S_2 for grading.

A more complicated example involves mutually suspicious subsystems. Suppose S_1 and S_2 own respectively subsystems T_1 and T_2; T_1 and T_2 are to cooperate, but neither trusts the other. For example, T_1 might be a proprietary subroutine which S_1 wants executed, but not read, by others; and T_2 might be a data management system which accesses certain files owned by S_2. Figure 7 shows how the cooperation can be arranged. Observe that T_1 may access only the objects of S_2 (such as X_2) which are accessible to T_2, but no others, and that S_2 may revoke T_1's access to T_2 at any time. (The converse is true for T_2 and S_1.) Observe that T_1 can only use, but not acquire for itself, access attributes of T_2.

We have now developed the necessary basis for a protection theory, allowing us to view the protection state of a system as being defined dynamically by its access matrix. We have shown how this model allows for protection at the fifth level discussed in the introduction. The utility of the abstractions, both in understanding current protection systems and in formulating future "ideal" protection systems will be discussed shortly, after we have considered the notion of correctness of a protection system.

CORRECTNESS, SHARING, AND TRUST

To prove that a protection model, or an implementation of it, is correct, one must show that a subject can never access an object except in an authorized manner. Two things must be proved: any action by a subject which does not change the protection state cannot be an unauthorized access; and any action by a subject which does change the protection state cannot lead to a new protection state in which some subject has unauthorized access to some object.

With respect to the first, given the correctness of each monitor, it follows that the attachment by the system of the identification number of a calling subject to each reference makes it impossible for a subject to gain unauthorized access to an object. The most important requirement in the argument is thus the assumption that the system attaches a nonforgeable identification number to each attempted access. It is necessary to prove that the operating system attaches the identification number correctly, the monitors interrogate the correct entry in the access matrix, and no monitor (except the access matrix monitor) alters the contents of the access matrix.

With respect to the second, it is clear that each application of rules R1-R8 produces a new protection state in the manner specified, assuming the correctness of the access matrix monitor. There are, however, various aspects of trust implicit in rules R1-R8. Specifically, if the owner of a given object is not careful, it is possible for untrustworthy subjects, acting singly or in collusion, to grant some subject access to that object, access not intended by the object's owner. Since the model cannot deal satisfactorily with problems relating to trust, this will demonstrate the need for external regulations or laws to complement the technical solution implemented by the model. We shall explore this point in the paragraphs following.

Consider the transferring and granting rules (R1 and R2). Suppose subject S_1 has created an object X and granted various types of access for X to subjects $S_2, \ldots, S_n$. Suppose further that S_1 intends that, under no circumstances, should S_0 gain access to X. He can do this by avoiding granting any access attributes to S_0, but he must also grant attributes with the copy flags off to any subject among $S_2, \ldots, S_n$ who he feels might violate trust and grant access for X to S_0 indirectly. In other words, an understanding must exist between S_1 and subjects receiving access to X that S_0 is not to receive, either directly or indirectly, any access to X. Only when these considerations are understood and the requisite trust is present should S_1 be willing to pass a copyable attribute. With this, rules R1 and R2 are correct.

On the basis of the foregoing argument, we conclude that the protection system is correct and will operate exactly as intended among *trustworthy* subjects. Untrustworthy subjects cannot be dealt with completely by mechanisms of the protection system. External regulation, together with a system for detecting and reporting violations, is required.

Consider the subject creation rule (R7). The correctness of the environment of the created subject is founded on the assumption that the creator's environment is correct, since a subordinate subject cannot be initialized with any access attribute not held by its creator. Thus the correctness of any subject's environment derives ultimately from the universal subject's having created a correct environment for a user when he signs on the system. Here is where the problem of false identification enters. An arbitrarily complicated question-and-answer procedure can be developed to

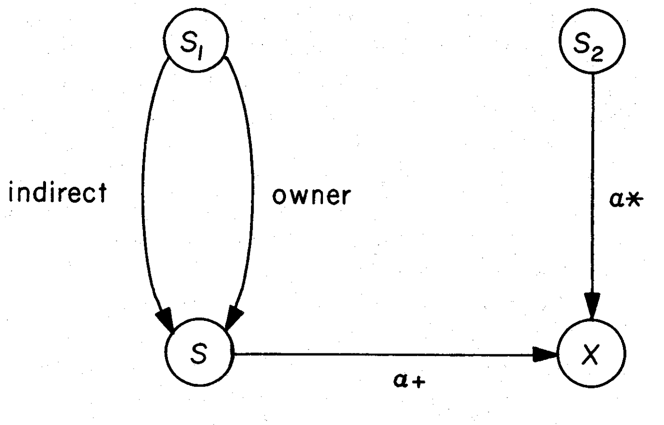


Figure 8—Preventing unwanted indirect access

establish the identity of a user to within any reasonable doubt, but it cannot be done absolutely. Again, a system for detecting and reporting violations of this type is required, and external regulations must deal with offenders.

As further illustration of the role of trust in the model, we consider four instances of trust's being implicit in the interpretation of access attributes themselves. First, consider the 'read' attribute. Reading is an unusually powerful operation, as it implies, for example, the ability to read and copy a file. It is then possible for one subject to distribute access attributes for the copy freely to other subjects, even to subjects who were not authorized to read the original. In this sense, the 'read' attribute is equivalent to the 'read*' attribute.

Second, consider the 'indirect' attribute, and refer to Figure 8. Subject S_2 has α^* access to X and wishes to transfer α to S. However, the owner S_1 of S may grant itself 'indirect' access to S and so gain α access to X, even though this may not have been intended by S_2. There are several ways to solve this:

(1) Define an access mode, denoted by the symbol '+,' which cannot be used in an indirect manner; thus $\alpha+$ is not only nontransferable, but it is usable only by the subject to whom it was granted.† This is shown in Figure 8.

(2) Do not allow an owner to grant itself 'indirect' access to a subordinate subject.

(3) Make α a "limited-use" attribute so that the exposure of X to the owner of S is of limited duration. It is not clear that (2) or (3) are viable solutions.

Third, consider the 'owner' attribute. A subject creating an object is given 'owner' access to that object, indicating that it may grant access attributes for that object to other subjects. It is possible in our model for the 'owner' attribute of a nonsubject object to be transferred, and thus multiple ownership may arise. Prior arrangements must exist among the owners, else contradictory actions may result—e.g., one owner grants access the others do not want granted. Such difficulties can be avoided by allowing only one owner, e.g., 'owner' is either untransferable or transfer-only but it is not clear whether this is a desirable or useful solution.

Fourth, consider rule R3, which permits the owner of an object to revoke arbitrarily any access held by any nonsubordinate subject to that object. This ability is open to question. Vanderbilt argues that no such ability is warranted as, in effect, the owner of an object and those to whom access has been granted have entered a contract[19]—i.e., the nonowners presumably have used the object in their programs and depend on its presence. Lampson, on the other hand, argues that, for the proper implementation of cooperation among mutually suspicious subsystems, absolute power of revocation is warranted.[16] The latter view is reflected in our model.

These considerations illustrate that trust is a distinctly nontechnical concept. They reemphasize an earlier point, viz., the complete solution to the protection problem must strike a balance between technical and nontechnical issues. More research is required on both issues, especially on methods of detecting and reporting violations and on coordinating external regulations with internal protection mechanisms.

Two other ways of breaching the security of protection systems exist, but do not fall in the province of the model. First is the "wire-tapping" issue. We consider this a solved problem, since coding techniques exist that render it essentially impossib!e for anyone tapping a data line to decode what he finds there. Second is the "system-error" issue. What if a system error changes the access matrix or the identification number of a subject or other object? Can a subject gain access it did not have before the error? The problem can be solved by appropriate use of coding techniques for subject and object identification numbers and for access attributes; in particular, it is possible to use error-correcting codes, and to arrange that the probability of an identification number (or access attribute) being changed to another valid identification number (or access attribute) is arbitrarily small.

† We have defined altogether four modes for access attributes: α^*, $\alpha\cdot$, α, and α^+, where α^+ is the most restricted mode of α and α^* the least.

SYSTEM IMPLICATIONS

The protection model presented above does not resemble any particular system with which the reader is familiar, although it likely contains concepts with which he is. By considering implementations of the access matrix, one can show that many protection systems are practical examples of the theory.

Storing the access matrix

Since the access matrix is likely to be sparse, it would be impractical to implement it directly as a two-dimensional matrix. A more efficient implementation could store the access matrix as a table of triples $(S, X, A[S, X])$—i.e., the nonempty entries of A. Since most subjects and objects would not be active at any given time, it would be unnecessary to store all the triples in fast memory at once. Moreover, it is often necessary in practice to be able to determine what object a given subject can access (or what subject can access a given object); a simple table of triples has too little structure to permit efficient search procedures to be implemented. Therefore, this method is unlikely to be practical, especially when the number of subjects and objects is large.

There are, however, at least three practical implementations. The first uses the idea of storing the access matrix A by rows, i.e., with each subject S is associated a list of pairs $(X, A[S, X])$, each pair being called a *capability* and the list a *capability list* (C-list).[4] A C-list represents all the objects that a subject can access, together with the authorized modes of access. Following this approach, one may regard a C-list as defining the environment (domain) of a process, a subject as a pair (process, C-list), and the operation of switching domains as switching to a new C-list.[4,16] Because the protection system allows only authorized operations on C-lists, possession of a capability by a process is *prima facie* proof that the process has access to an object.[15]

A second approach uses the idea of storing the access matrix by columns, i.e., with each object X is associated a list of pairs $(S, A[S, X])$. In Multics, such a list is called an *access control list* when the objects are segments;[1,18] in the Cambridge multiple-access system, it is called an *authority list.*[20]

A third approach represents a compromise between the C-list and the access-control list implementations. Suppose a set of subjects desires access to a certain set of objects. In the C-list of each subject will appear entries (X, K) where X is an object name and K a *key.* Associated with the set of objects is a *lock list* containing entries of the form (L, α) where L is a *lock* and α an access attribute. The monitor of the set of objects, upon detecting that S is attempting to access X in manner β, will obtain an (X, K)-pair from the C-list of X; it then will search the lock list and permit the access only if it finds an (L, α)-pair for which $K=L$ and $\alpha=\beta$ (i.e., the key fits a lock that, when opened, releases the desired access attribute). In this system, the owner of an object X can distribute α access to X simply by granting (X, K) pairs to various subjects and placing (K, α) in the lock list. He can revoke this instance of α access to X simply by deleting (K, α) from the lock list (i.e., changing the lock); in this case the outstanding (X, K) pairs will be invalidated. It is possible for an owner to create various keys and locks for the same access attribute α—e.g., he can grant (X, K_1) to some subjects and (X, K_2) to others, placing both (K_1, α) and (K_2, α) in the lock list. This resembles the system of "storage keys" used in IBM System/360[12] with one important difference—viz., there is no concept of a C-list and it is possible for certain subjects to forge storage keys.

Efficiency

The viability of the model in practice will depend on the efficiency of its implementation. With respect to this, a few observations can be made. (1) Access attributes would not be represented as strings. An entry $A[S, X]$ of the access matrix might be represented as a binary word whose ith bit is 1 if and only if the ith access attribute is present. A corresponding binary word could be used to store the copy flags. (2) According to the rules, there is nothing to prevent one subject from granting another unwanted or unusable access attributes. Although this does not alter the correctness of the model, a practical implementation may require some mechanism to limit this effect. (3) The identification number of a subject can be stored in a protected processor register where it is quickly accessible to object monitors. (4) The implementation of the rules for creating and deleting columns and rows of the access matrix must be tailored to the manner in which the matrix is stored. In the C-list implementation (storage by rows), for example, deleting a row corresponds to deleting a C-list, but deleting a column would imply a search of all C-lists. In this case, it would be simpler to invalidate the name of the object whose column is deleted and use a garbage collection scheme to remove invalid capabilities from C-lists. (5) The C-list implementation is inherently more efficient than the access control list implementation because a C-list must be stored in fast-access memory whenever a subject is active whereas an object may not be active often enough to warrant keeping its access control list in fast-access memory at all times.

It is worth a special note that the mapping tables of a virtual memory system are a highly efficient form of the C-list implementation for memory protection. A separate table is associated with each process, and each entry of such a table is of the form (X, Y, α) where X is the name of a segment or page, Y is its location in main memory, and α specifies the types of access permitted to X (typically, α is some combination of 'read,' 'write' and 'execute'). In fact, the C-list proposal by Dennis and Van Horn[4] was motivated by generalizing the idea of mapping tables to include entries for objects other than segments. The point is: the same techniques used to make virtual memory efficient can be generalized to make the C-list implementation efficient. The requisite hardware has been described by Fabry[6] and Wilkes.[20] It is interesting to note that Multics, even though it centers on the access control list implementation, uses a form of the C-list implementation (the "descriptor segment") as well, the appropriate entries of the access control list being copied into the descriptor segment as needed.[1]

Choosing subjects and objects

The model assumes that an access attribute applies uniformly to an object and all its components, but allows components of objects to be objects themselves. It is possible, for example, to regard a file as an object and some of its records as objects too. Thus the designer has great flexibility in the choice of objects.

By taking a subject to be a (process, domain) pair, the model forces all components of a process to have identical access privileges to objects. Thus, whenever it is necessary to grant components of a process differing access privileges, it is necessary to define different subjects for each component. To illustrate this point, suppose a process P operating in domain D consists of subprocesses $P_1, \ldots, P_n$ which must be accorded differing access powers. Since the domain defines the access powers of a process, the subjects $(P_1, D), \ldots, (P_n, D)$ would have identical access privileges. Instead, the system would have to allow the definition of subjects $(P, D_1), \ldots, (P, D_n)$ where domain D_i is effective while subprocess P_i is executing; only then can the subprocess P_i of P be accorded differing access privileges.

As further illustration of the latter problem, consider a system implementing segmentation, in which a process P is uniquely and permanently associated with a name space described by a segment table T; i.e., each subject is a pair (P, T). Suppose $T=[(s_1, \alpha_1), \ldots, (s_n, \alpha_n)]$ where s_i is the ith segment and α_i defines the access P has to s_i. If now we want to debug a new segment s_{n+1}, there is no way to change P's access power when it enters segment s_{n+1}. In terms of Figure 6, it is not possible in this system to put the segment being debugged at a subordinate level. One solution would be to place s_{n+1} in its own name space with segment table T' and process P', so that the subject (P', T') being debugged is subordinated to the debugger (P, T). This cannot be done very efficiently, as software-implemented message-transmitting facilities would have to be used for communicating between the processes P and P'. The problem could have been avoided in the first place if it were possible to define a series of domains $T_i=[(s_i, \alpha_i)]$ and subjects $S_i=(P, T_i)$ so that S_{n+1} can be made subordinate to $S_1, \ldots, S_n$. This is the approach suggested by Evans and Leclerc.[5]

Existing systems

We examine now several important systems in the light of the model. For each system we will identify the elements of the model, points of inefficiency, and points of insecurity (if they exist).

One of the most common systems in use is IBM's OS/360.[11] In this system, objects typically are files, tasks, and instructions. The domains correspond to pairs (mode, block) where "mode" refers to the supervisor/problem-program machine states and "block" refers to a block of the multiprogramming partition of main memory. A subject is then a triple (task, mode, block). One of the principal elements of the model is not present, viz., the idea of attaching a subject identifier to each attempted access; the storage-key mechanism in effect does this for access to objects in memory, but there is no such notion associated with attempts to switch domains or to reference files. It is possible for a task to guess a supervisor call number and issue a supervisor call (SVC) with that number, thereby gaining unauthorized access to the supervisor domain. It is also possible for a user to guess a file name and access it through the job control language.

OS/360 is representative of many current commercial systems in the lack of an adequate protection system and underlying protection theory. The limitations above indicate the OS/360 does not even provide protection at the lowest level mentioned in the Introduction, i.e., it cannot guarantee absolutely that each user is protected from the ravages of other users. The two systems described below (RC4000 and Multics) do embrace the concepts of the model, concepts which must be embraced widely in the commercial systems before they will be capable of providing protection at the level required for sharing mutually suspicious subsystems. There are, of course, other systems which embrace the concepts of the model, but which cannot be discussed here for lack of space; these include the CAL system at Berkeley,[14] the capability hardware proposed by Fabry at Chicago[6] and Wilkes,[20] Project SUE at the University of Toronto,[8] and the IDA system.[7]

The RC4000 system is one example of a commercial system having a protection system incorporating all the elements of the model.[2] The objects of this system include the usual repertoire of files, devices, instructions, and processes. In this system, each process defines a domain (i.e., subjects are identical to processes), and a "father-son" relation defines a hierarchy of processes. Each object is associated with exactly one domain; therefore each process has exclusive control over certain objects, and it must request other processes to access other objects on its behalf. (In this sense, each process is the monitor of its objects.) For this purpose, the system implements a message-transmission facility and associates with each process a message buffer in which are collected all messages from other processes. A process P wishing to send message α to process P' uses a system call

send message (α, P'),

the effect of which is to place the pair (P, α) in the message buffer of P'. Note that this is precisely the notion of attaching the identification number of a subject to an attempted access, if we take the view that each process is an object monitor and a message is an attempted access to some object controlled by the monitor. This system provides a great deal of flexibility: each process plays the role of a monitor; the number of monitors is variable; the definition of an object can be dynamic since the programming of a process can be used to implement "virtual objects" which are accessible to other processes via the message-transmission facility; and each process can determine the authenticity of each message it finds in its message buffer. The limitations of this system include: the message facility, being implemented in software, is of limited efficiency; there is no way to stop a runaway process, as the system cannot force a process to look in its message buffer; and a possibly elaborate system of conventions may need to be adopted in order that receivers can learn the identification numbers of senders. Further discussions of these points are found in References 3 and 15.

The Multics system at MIT is an example of a noncommercial system which implements the elements of the protection model. Typical objects are segments and processes. The virtual memory is the monitor of the segments and the traffic-controller the monitor of the processes. The domains of this system are defined by pairs (ring, base) where "ring" refers to integers 0, 1, 2, ... and "base" is a pointer to the descriptor segment (segment table) of the name space. Each process is associated with exactly one name space so the base can be used as its identification number. Each segment s has associated with it a ring number n_s; if process P is executing instructions from segment s in name space with base b, the subject is the triple (P, n_s, b). Observe that a control transfer from segment s to segment t for which $n_s \neq n_t$ defines a change of subject. The access privileges of a subject are considered to diminish as its ring number increases; in other words, for given P and b, the ring numbers 0, 1, 2, ... define a linear hierarchy of subjects $(P, 0, b)$, $(P, 1, b)$, $(P, 2, b)$, ... Associated with segment s are three pairs of integers

(r_1, r_2)—the read bracket, $r_1 \leq r_2$

(w_1, w_2)—the write bracket, $w_1 \leq w_2$

(e_1, e_2)—the execute bracket, $e_1 \leq e_2$

(These numbers can be stored in the segment table along with flags indicating whether each type of access is permitted at all. In Multics, only three of the seven integers associated with a segment are distinct, so only three integers need be stored to represent the three access brackets and ring number.) In terms of a subject hierarchy, the notion of an access bracket for access attribute α of an object means simply, for a certain range of levels above and below the owner of the given object, a superior or subordinate subject within the range is automatically granted α access to the object. It is an extension of the concept of access attribute. The access matrix is stored statically in the form of access control lists associated with objects (segments). This information is copied dynamically into the descriptor segment as segments are referenced for the first time, so that the descriptor segment is a form of a capability list. A subject (P, n_s, b) may access in any manner any segment t listed in the descriptor segment when $n_t > n_s$; furthermore, if $n_t \leq n_s$ it may read, write, or execute (transfer control to) t if n_s falls in the proper attribute access bracket of segment t. If it wishes to transfer control to a segment t whose execute bracket is (e_1, e_2) but $e_2 < n_s$, it may do so by transferring to a "gate" (protected entry point) of segment t; the attempt to do so generates an interrupt which invokes a "gatekeeper" that permits transfers only to gates. A full description of the mechanism as outlined above can be found in References 9 and 18.

CONCLUSIONS

We have carefully tried to limit the complexity of the model, omitting from its basic structure any feature which is either controversial or not completely understood. It is possible to extend the model in any or all of the following ways. (1) A subject which creates an object can specify that the object is permanent, so that the universal subject can make that object available for future reincarnations of that subject. Alternatively, a subject can specify that, if it dies, ownership of objects it owns can revert to a superior subject. (2) In a subject hierarchy, superior subjects can be regarded as dormant when their processes are not executing, so that subordinates can send messages which, for example, request additional access attributes. (3) Since a subordinate subject may acquire access attributes not held by its superiors, rules may be required specifying when superiors can acquire such attributes for themselves; Lampson allows for this.[15] (4) Given the possibility of multiply-owned objects, rules may be required to permit owners to guard their access attributes from deletion by other owners. (5) If one adopts the view that a subject and all its subordinates constitute a "family," [2] one can make a case for allowing a subject to transfer an access attribute to a subordinate even if the copy flag is not set.

The considerations of the section on System Implications indicate that most current commercial systems do not implement a complete protection mechanism, even at the lowest level described in the Introduction. The most serious defect is the lack of completeness when a process changes protection environments. Although the incompleteness of these systems could be rectified by (probably extensive) software modifications, a complete protection system cannot hope to be efficient unless the basic elements of the model are implemented in hardware. The requisite hardware has been described in many forms—by Fabry,[6] by Schroeder and Saltzer,[18] and by Wilkes[20]—and is in fact quite straightforward.

It can be noted that the "average" user, who employs only system compilers and library routines, is unlikely to break the security of the system if only because compilers and library routines will not cause the loopholes to be exercised. This does not mean attention should not be paid to structuring the hardware so that even the most enterprising and malicious user cannot break security. It takes only one such user to compromise another's privacy which, once lost, may be irrecoverable. On this basis, the cost of hardware and software development required to achieve efficient implementations of protection is of the utmost value.

We began the paper with the statement that the technical approach to protection had not been treated adequately to date in the literature. Hopefully, the discussion here has shown that it is possible to state the problem on an abstract level where the elements of protection can be isolated, where methods of proving the correctness of a protection system can be formulated, where drastically different physical implementations of the one model can be compared and evaluated, and where the nontechnical issues required to complement the technical ones can be identified. The discussion here has been intended as an example of what is possible in a model; we are under no delusions that this is the only model or that this is the best model. Our preliminary work has indicated that the abstractions formed in the modeling process are useful in themselves and that the model provides a framework in which to formulate precisely previously vague questions. We hope that this discussion will motivate others to undertake additional research in this area. Much needs to be done.

ACKNOWLEDGMENTS

We are indebted to Michael Schroeder for pointing out the seven levels of sharing mentioned in the Introduction, and to Richard Holt, Butler Lampson, Richard Muntz, and Dennis Tsichritzis for many helpful discussions and insights.

REFERENCES

1 A BENSOUSSAN C T CLINGEN R C DALEY
The MULTICS virtual memory
Proc 2nd ACM Symposium on Operating Systems Principles Oct 1969 pp 30-42

2 P BRINCH-HANSEN (Ed)
RC-4000 software multiprogramming system
A/S Regnecentralen Copenhagen April 1969

3 P J DENNING
Third generation computer systems
Computing Surveys 3 4 Dec 1971

4 J B DENNIS E C VAN HORN
Programming semantics for multiprogrammed computations
Comm ACM 9 3 March 1966 pp 143-155

5 D C EVANS J Y LECLERC
Address mapping and the control of access in an interactive computer
AFIPS Conf Proc 30 Spring Joint Computer Conference 1967 pp 23-30

6 R S FABRY
Preliminary description of a supervisor for a machine oriented around capabilities
ICR Quarterly Report 18 University of Chicago August 1968 Section I pp 1-97

7 R S GAINES
An operating system based on the concept of a supervisory computer
Comm ACM 15 3 March 1972

8 G S GRAHAM
Protection structures in operating systems
MSc Thesis Department of Computer Science University of Toronto August 1971

9 R M GRAHAM
Protection in an information processing utility
Comm ACM 11 5 May 1968 pp 365-369

10 L J HOFFMAN
Computers and privacy: a survey
Computing Surveys 1 2 June 1969 pp 85-104

11 *IBM System/360 operating system concepts and facilities*
IBM Report No GC28-6535 November 1968

12 *IBM System/360 principles of operation*
IBM Report No GA22-6821 September 1968

13 *IBM System/370 principles of operation*
IBM Report No GA22-7000 June 1970

14 B W LAMPSON
On reliable and extendable operating systems
Techniques in software engineering NATO Science Committee Working Material Vol II September 1969

15 B W LAMPSON
Dynamic protection structures
AFIPS Conf Proc 35 Fall Joint Computer Conference 1969 pp 27-38

16 B W LAMPSON
Protection
Proc Fifth Annual Princeton Conference on Information Sciences and Systems Department of Electrical Engineering Princeton University Princeton New Jersey 08540 March 1971 pp 437-443

17 S MOTOBAYASHI T MASUDA N TAKAHASHI
The Hitac 5020 time sharing system
Proc 24th ACM Nat'l Conference 1969 pp 419-429

18 M D SCHROEDER J H SALTZER
A hardware architecture for implementing protection rings
Comm ACM 15 3 March 1972

19 D H VANDERBILT
Controlled information sharing in a computer utility
MIT Project MAC report MAC-TR-67 October 1969

20 M V WILKES
Time sharing computer systems
American Elsevier 1968

PRIME—A modular architecture for terminal-oriented systems*

by HERBERT B. BASKIN, BARRY R. BORGERSON and ROGER ROBERTS

University of California
Berkeley, California

INTRODUCTION

The architecture of most interactive systems is based on the general strategy that suitable terminal service can be provided by a central processor that is time-multiplexed among all the active terminals. In order to achieve adequate response time in an interactive environment, the CPU is usually time sliced. Other major system facilities such as I/O channels and secondary storage units are also shared among the users, and multiprogramming techniques are employed to keep all the major system resources as fully utilized as possible. An operating system is usually developed which performs these functions as well as supervising the terminal communications, implementing a system-wide filing subsystem, handling user commands, etc. The result of combining these and other functions into a time-sharing operating system is a highly complex software system which transforms what is basically a batch processing computer structure into a multi-terminal system with significant limitations that are an outgrowth of this strategy. While a failure can occur in any section of the hardware or software, we know that hardware failures are more likely to occur in the electromechanical and core memory sectors than in solid state logic, and that software failures tend to be concentrated in the more complex areas of code. Failures of hardware components may require modification of the operating system in order to regain operational status since the allocation strategies may need more than parametric modification when system resources are affected.

The main attribute of a computing facility that will serve many users simultaneously is that it provide continuous service. By this we mean the ability to service as many users as possible even when a failure has occurred within the system. Failures must be considered normal occurrences which can co-exist with a full range of other normal operations such as resource allocation, interpreting user commands, etc. However, conventional time-sharing systems cannot cope with these failures, and hence cannot offer continuous availability, since large and complex portions of both hardware and software are concentrated in unique units of the system. While it may appear on the surface that further development and refinement of the basic strategy will bring one closer to the goal of continuous availability, we feel this is a false evolutionary path. Instead the approach we have taken is to begin with a totally different architecture which achieves these goals at the very outset.

All of the above considerations are no less true when several processing units are employed as scheduling processors, communications processors, file control processors, etc. The basic problem is that such organizations are all predicated upon the notion that the normal mode of operation of the overall system is when no failures exist. In any practical utility, such as a telephone system or power system, it is always assumed that failures exist as a normal occurrence and must be treated while continuing as near normal operation as possible. This assumption is one of the basic cornerstones of the architecture described in this paper.

Another motivation behind our approach is to provide the ability for any user to continue undisturbed with his use of the system even though it is undergoing considerable change. In a conventional architecture, all user programs run under system-wide software which, if changed, may require the user to update his programs. Even when a considerable effort is made to

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC15 70 C 0274. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

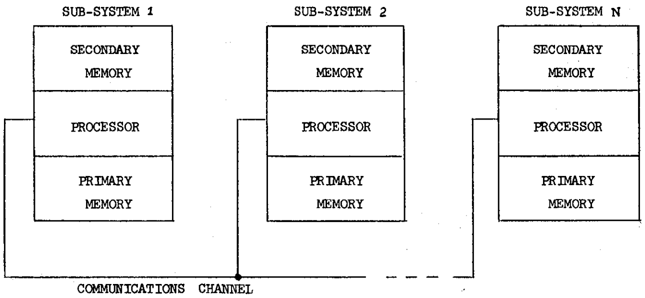


Figure 1—Canonical system

avoid such situations, they do occur, and represent a fundamental flaw in the design. In the PRIME system, the only requirement is that user software be compatible with the processor instruction set.

A user who wishes to have a local operating system supplied to him may avail himself of a currently available version, use a colleague's system, or supply his own. As a result of this new degree of freedom the user may elect to continue using a version of an operating system which has been superseded, thus insuring that his programs, which have been running in a satisfactory manner, will continue to do so. This is of considerable importance in a commercial environment.

SYSTEM ARCHITECTURE

The PRIME system structure is highly *modular* in that it is composed of a small set of different functional units, each of which is replicated many times. The major functional units include primary memory modules, secondary storage modules, and processing modules, which are so interconnected that larger functional units are obtained, each a replication of another. An understanding of this architecture can best be gained by examining the structure and properties of a conceptual system which we call the *canonical system.* As can be seen from Figure 1, the canonical system is composed of a number of subsystems, each independent of the others, which can communicate with each other through a communications channel. Since a subsystem is composed of a processor, primary memory, and secondary storage, it is completely self-contained and able to function independently. The communications channel is used to facilitate data transfers between subsystems. It can be used only in a very specific manner, in that both the sender and the receiver of the data must agree to set up the communications path, and is present mainly for data base sharing operations among users.

This canonical system consists of n identical subsystems which can process n independent jobs with an extremely high degree of protection from each other. Software faults within a subsystem, whether in the user software or in the local operating system (within each subsystem) cannot affect the other subsystems. In addition, hardware failures for the most part only affect the faulty subsystem. There are, however, some possibilities that certain hardware failures (in the communications channel) can cause erroneous interconnections between the subsystems. In these cases, which are strictly non-electromechanical, fault detection schemes can be provided which would guarantee the integrity of the communications channel. System availability with this approach is therefore very high since a failure (hardware or software) usually affects very few of the subsystems.

The canonical system, as was shown above, could provide us with a highly secure and available system. However, in order to achieve more efficient and useful system operation, it is necessary to have methods for allocating the system resources among a large number of users. This must be accomplished without degrading the security or availability characteristics of the canonical system. Accordingly, three additions are made to the canonical system structure just described. The first addition is to place a memory switching unit in each path from a processing module to its primary memory modules, and to allow each memory switching unit to connect to a large number of primary memory modules. This allows processing modules, and therefore basic subsystems, to share primary memory modules and therefore allows for variable primary memory resource sharing. The second addition is to place a switching unit in each path from a processing module to its secondary storage modules. This modification, together with the previous one, allows for extensive resource sharing within the system. The third addition which is necessary to manage allocation of system components is a controlling process, called the *control monitor.*

The resultant system now has facilities for extensive resource allocation, but still retains the security and availability properties of the basic canonical system. This is because, at any instant in time, PRIME is configured to be n subsystems, each consisting of a processor, primary memory, and secondary storage. Each of these subsystems can be considered to be logically and electrically distinct, and as such, compose a system which is essentially similar in structure to the canonical system. The effect of the additions was merely to allow the specific partitioning of the configurations to vary over time. In order to achieve the capability to allocate and repartition the system we use one subsystem to implement the control monitor.

Its processor is designated the *control processor* while the remaining processors are called *problem processors.* Since the assignment of roles of the processors is completely arbitrary, it can vary on a dynamic basis. The functions of the control monitor include job scheduling, terminal buffer management, secondary storage allocation, primary memory allocation, low-level accounting, system diagnosis and problem processor monitoring. The problem processors, on the other hand, have as their main responsibility that of executing user jobs. In order to accomplish user service functions, each problem processor contains a *local monitor,* whose functions include user program file transfers, user program trap routines, and general program I/O.

Each of the subsystems is completely self-contained in that each processing module is composed of a number of functional units which allow it to fully complete a processing cycle. By this we mean that once a subsystem composition is formed and a user process has been activated, no further interaction between that process and the control monitor is necessary until the completion of the job step. Additionally, each processing module contains functional units, inaccessible to user processes, which are used for system-wide functions and control. The particular functional units implemented in each processing module are:

(1) target machine
(2) disk control
(3) memory access
(4) terminal control
(5) communications channel
(6) control monitor extension

where the first three are for user process execution and the last three are system functions.

The target machine functional unit executes machine language instructions and is the interface between user software and the system. A complete description of it would be beyond the scope of this paper.[2]

The memory access and disk control functional units allow the processing module to access, in a secure manner, primary memory modules and secondary storage modules. Each is composed of machinery restricting these accesses to allocated modules. This latter machinery is in the form of maps, which translate user virtual addresses into system absolute addresses. The contents of the maps are set by the control monitor, in a manner described below, at the inception of a job step, when the control monitor allocates primary memory and secondary storage modules to the process just starting up.

The terminal control functional unit in each processing module is connected to a subset of all the terminals connected to the system, and transfers characters between them and the control monitor. With this organization, we have obviated the need for a unique multiplexor, with its attendant reliability problems, and have distributed the terminal control throughout the system. It should be noted that there is no direct relationship between the terminals connected to a processing module and the user processes executing on it.

The information flow between the control monitor and processing modules, as discussed above, is accomplished through the agency of the communications channel functional unit in each processing module. This unit allows for data to be exchanged between the control processor and any of the problem processors.

At the heart of each processing module, managing the activities of the previously discussed functional units, is the control monitor extension functional unit. Each ECM (extended control monitor) is actually a part of the control monitor, performing local actions on its behalf within each subsystem. These actions include setting and reading the map entries of the memory access and disk control functional units, saving and restoring the state of user processes, managing the swapping of user processes, interfacing between the control monitor and user processes, and implementing local error checking and diagnostic algorithms. It should again be noted that the ECM is not implemented in software, executing on the target machine functional unit, but is a distinct functional unit completely inaccessible to and independent of user programs.

The combination of the control monitor executing on the control processor, and each of the ECMs contained within problem processors provides for a rational distribution of responsibility and a concomitant decrease in complexity. Global allocation of primary memory modules and secondary storage modules to processes, and processes to processors is performed by the control monitor, while local allocation of virtual addresses to absolute addresses and local process control is performed by each ECM. Similarly, global error checking and diagnosis are performed by the control monitor, while local error and consistency checks are performed by each ECM.

The main result of this organization is that system functions are completely shielded from user programs in that the control monitor and user processes are never executed on the same processor. There is a physical barrier, brought about by the structure of the system, between those system operations which must remain secure from interference by potentially harmful user software. The allocation of the system between system functions and user functions is not time-multiplexed, as in most conventional organizations, but rather distributed in space. This has many implications relating to system security. The two most important are that since the control monitor is always executing, it is able to continuously monitor the state of the system and that a failure, either in hardware or software, does not cause a breach in the barrier between the system and a user. The decrease in complexity of the control monitor which accrues from the fact that it runs on an independent processor and is involved with each active process only at its initiation also contributes to system security and availability. With a less complex monitor, it is considerably easier to prove that the system software, as it relates to security functions, is operating correctly. Additionally, since the control monitor is responsible for only a subset of the system functions, and hence maintains only a few of the system state tables, it is decidedly easier to recover from a control processor failure.

The processing modules, as described, consist of six independent functional units. In the first implementation of PRIME, however, these processing modules will be constructed from microprocessors and a modest amount of special hardware, with the separate functional units being micro-code subroutines. We feel that this organization is desirable since we are developing a system with which we can conduct experiments with different operating procedures and system organizations. We consider the first implementation of the PRIME system to be a laboratory within which we can experimentally implement a variety of different systems that have in common a number of the architectural features described above.

The decision to employ microprocessors was made to allow for maximum flexibility of the system for such experimentation. An alternate implementation choice would be to construct each processor module from a number of very small inexpensive processors. In addition to the flexibility advantage, the microprocessor choice currently has a cost advantage. However, this situation is expected to radically reverse itself in the next few years. The PRIME architecture will allow us to keep up with the trend toward low cost LSI processors without any architectural changes to the system since later expansion would be with additional modules that are functionally similar to the microprocessor modules, but are internally implemented using several LSI processing units.

SYSTEM COMPONENTS AND OPERATION

The components of the PRIME system are interconnected in the manner illustrated in Figure 2. The processing modules are general-purpose micro-programmable processors which operate on 16-bit operands with a cycle time of approximately 100ns. The control storage for these processors is arranged in 32-bit words, which is also the instruction width. When fetching operands from control store, only half of the word is used. The processor has 2048 words (32-bit wide) of control store and 32 directly addressable 16-bit registers. The data processing logic consists of an arithmetic/logical unit which processes data received from two source buses (A and B buses) and stores the result into the destination register via a third bus (D-bus) after passing the data through a shifter unit. The processor we use which has these characteristics is the META 4 microprocessor units supplied by Digital Scientific Corp.

The processing element uses 21 double bus accumulators and eight general-purpose I/O registers. Five of the I/O registers are used to drive the primary memory system. Of these, two are for transferring data, one for the operand address, one for the instruction address, and one for the memory status. In each processor, the remaining three I/O registers drive the I/O bus which is used for all communications between that processor and external devices. This includes devices attached to the I/O terminals, the real-time clock, the time-of-day clock, the memory map, fault detection messages, and the external access network which will be described below.

The primary memory system for each processing module consists of the five I/O registers, a memory interface which controls memory traffic and contains a memory map, and a memory bus. As shown in Figure

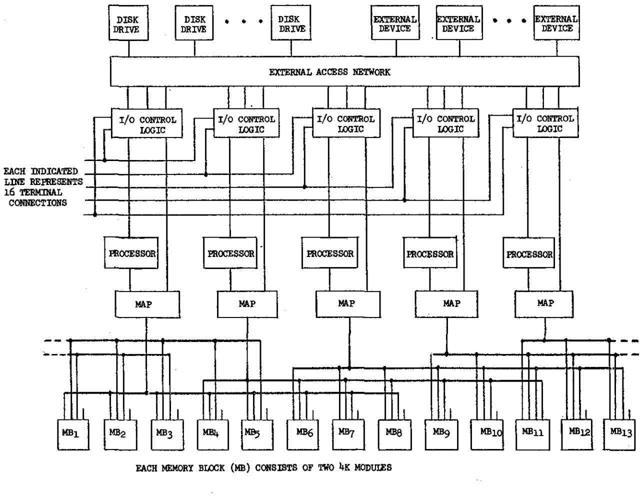


Figure 2—A block diagram of the PRIME system

3 each memory block consists of a four-by-two switching matrix and two 4k × 33 bit memory modules, each of which is partitioned into four 1k word pages.[3] There are five memory buses in the system, each of which connects a processor to eight memory blocks. Thus each processor connects to 16 of the 26 4k memory modules in the system.

Since 64k words of memory is substantially greater than the size of the working sets we anticipate encountering, the allocation of memory modules will be simple and straightforward. In addition, the possibility of being unable to schedule a process due to insufficient free memory modules will be remote. The use of the four-port memory switching units not only provides a distributed memory switch which contributes to the high availability of the system, but also allows for experimentation with different memory/processor connection algorithms.

With the exception of fault reporting and map loading and reading, the memory interface is independent of the I/O bus. The conflicts between the processor and the disks for the use of a memory bus are resolved in the memory control and switching unit shown in Figure 4. This unit also buffers a single word requested by the instruction address register, thus allowing instruction prefetches. In addition to translating logical pages into real pages, the memory map contains bits that specify whether a page is writable, readable, or executable. The use of separate operand and instruction address registers allows the hardware to determine the difference between an operand fetch and an instruction fetch and thus to easily detect when any reference violation has occurred. Other bits in the map include a dirty-page bit, a use bit, and a parity bit.

The memory is constructed from 1k bit MOS devices instead of the traditional core storage technology. Each 4k × 33 bit memory module is made up from 132 such chips. The memory modules have 400ns access and 550ns cycle times. Refresh is performed automatically within the module, using one cycle every 100 microseconds.

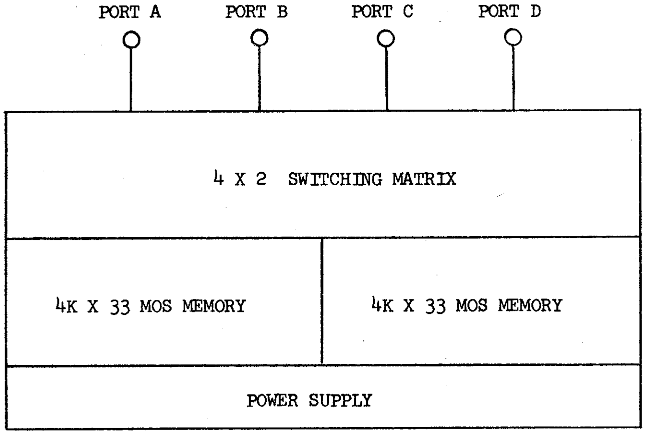


Figure 3—Memory block

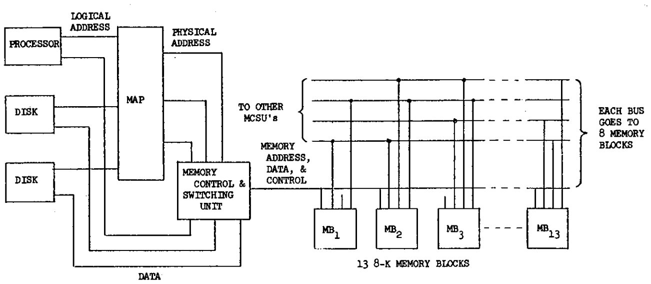


Figure 4—PRIME memory system

In the PRIME system, swapping space, secondary storage, and third level storage all reside on a single type of storage unit. This gives us another level of modularity and interchangeability which in turn contributes to the continuous availability of the system. The unique feature of the PRIME architecture which provides that each active process has its own processor and secondary storage unit makes possible this use of a single device for all of these functions. For this purpose we have chosen a commercially available high-performance moving-head disk drive, and have modified it to transfer data at twice its normal rate. This drive is a modified Model 213 unit supplied by Century Data. The basic Model 213 has 11 platters, 20 heads, 400 tracks, a capacity of approximately 60 million bytes, an average access time of 32 milliseconds, a latency of 12.5 milliseconds, and a transfer rate of 2.5 MHz. We have modified the units to read two heads in parallel so that at the interface it appears to have 10 heads and a transfer rate of 5MHz. Using two heads in parallel with moving head disks drives results in a deskewing problem which is especially acute when one tries to read data from a disk pack written on another drive. Rather than attempt to provide enough buffering to handle the worst possible case of head skewing, we chose to deskew the data by reading words from each head into every other memory location. That is, words assembled from one head are written into odd memory locations, and those from the other head into even memory locations. Thus if one head gets ahead of the other, the controller will simply wait for the last head to finish reading before proceeding. With this scheme we can roll in an 18k working set in under 200 milliseconds.

Each processor in the system can simultaneously connect to any two of the disk drives through the External Access Network as shown in Figure 2. One of these disk paths will normally be used for user file operations, and the other will be used for overlap swapping. Hence, if all ten disk drive paths are transferring simultaneously, the total transfer rate between secondary and primary memory will be 50 MHz or better than 6 million bytes per second. This is indeed a very high rate obtained from such inexpensive devices (approximately $.00003 per bit including modifications and controller). This type of high performance file capability is particularly important to interactive users as well as those who are data-base oriented. Such users tend to require turnaround times of ½ to 2 seconds within which they need only several tens of milliseconds of processing time. In the PRIME system, a process can be continuously active for up to several hundred milliseconds during which user paging activity could take place directly between a problem processor and its dedicated secondary storage unit. In such an environment, the amount of primary memory needed to efficiently execute a process is substantially less than that required in previous system structures.

With the exception of the terminals, all I/O between the PRIME system and external devices takes place via the External Access Network (EAN). This unit allows any processor to connect itself to any external device or to any other processor. Each processing module has three paths to the EAN. Each path contains a parallel interface which is 40 bits wide and transfers data asynchronously at a rate of about 100k words per second. On the processor end the data are routed directly to and from the processor by the I/O controller. In addition to the 40-bit path that communicates directly with the processor, each path through the EAN contains two lines that assemble data from the disk and transfer them directly to memory. These two bits are used only by the disk data paths. All other I/O, including that of the remote end of the disk controllers, takes place through the standard 40-bit path. Actually, the internal switching unit that is part of the EAN is only four bits wide in each direction. The 40 bits are always disassembled, transmitted through the switch, and then reassembled at the other end. However, these are internal details within the EAN and are not apparent at any of its ports. The cost of the switch within the EAN is a function of the product of the number of processors and external devices and also is proportional to the width of its internal data path. These considerations together with our bandwidth requirements have led us to the selection of a four-bit internal switching path. A more complete discussion of the design considerations of the EAN is presented in another paper in these proceedings.[4]

Each processing module is connected to an equal number of terminals, thereby achieving a distributed input/output capability. Each terminal is connected to two processors so it can continue to run in the presence of a failure of a processing module. This provides back-up at minimum cost since the connections to the processors consist simply of level shifting circuits. All assembly and disassembly of characters is done by the microprocessor on a bit by bit basis. As soon as a character is assembled by a processor, it is sent to the control processor where line editing and buffering takes place.

Interprocessor communication takes place via the External Access Network. Direct communication between problem processors is not allowed. Rather, all interprocessor communication is initiated by the control processor connecting itself to a problem processor through the EAN.

The External Access Network treats all of the processor ports in a symmetrical manner. However, at any point in time one processor port is designated as the controlling port and as such has the capability to establish a link between any other processor port (addressing port) and an addressable port to which external devices may be connected. Thus we not only achieve the above mentioned ability for a control processor to communicate with other processors and for any processor to connect to any external device, but by redesignating the controlling port, we can make any of the processors in the system the control processor. We have developed techniques for the redesignation of the control processor that would allow for continuous operation of the PRIME system in the presence of a hardware failure anywhere in the system. These techniques will be described in a later paper.

SUMMARY

We have described a system which is currently being constructed by the Computer Systems Research Project at the University of California at Berkeley with the support of the Advanced Research Projects Agency. Our principal goals are to open up new architectural avenues that will lead to simpler, more reliable, and secure terminal oriented systems. The architecture we have described here provides the user with a target machine having an exceptionally high bandwidth to secondary storage and a powerful medium scale computer on a dedicated basis during the period that he needs processing. The technique of having a bank of components to draw from to construct such a user subsystem allows for nearly continuous operation without having to use redundant components.

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions of a number of people who include Larry Barnes, Robert Fabry, Domenico Ferrari, Richard Freitas, Charles Grant, Mar, Greenberg, Paul Morton, Jesse Quatse, and C. V. Ravi.

REFERENCES

1 H BASKIN et al
A modular computer sharing system
Communications of the ACM 12 pp 551-559 October 1969

2 R ROBERTS
The target machine for the PRIME system
Internal document no. R-1.0 November 1971

3 C V RAVI
Memory addressing and processor-memory interconnection
Internal document no. W-13.0 August 1970

4 J QUATSE P GAULENE D DODGE
The external access network of a modular computer system
AFIPS Conference Proceedings Vol 40 Spring Joint Computer Conference 1972

Computer graphics terminals—A backward look

by CARL MACHOVER

Information Displays, Inc.
Mt. Kisco, New York

INTRODUCTION

Five years ago, the price of admission into Interactive Computer Graphics was spending about $50,000 or more for the Graphics Terminal and associated hardware, plus writing almost all of the applications software, as well as much of the basic software. And, only about a dozen suppliers offered commercial equipment.

Today, the price of admission has dropped dramatically. Graphics Terminals can be purchased for less than $10,000. Some turnkey applications software packages are available. And the buyer can choose from among more than 35 hardware and system suppliers, offering over 60 different models.

Some aspects of graphics terminal performance have not changed materially over the past five years. For example, maximum screen data content (number of flicker-free points, characters and lines) has remained almost constant for refresh displays. However, there have been significant advances in other areas, such as intelligent (minicomputer-based) terminals, low cost terminals, color displays, specialized hardware function devices, graphic tablets, and the use of storage tubes and digital TV to increase screen content.

SUPPLIERS

In my FJCC Paper about five years ago,[1] I listed sixteen manufacturers of commercially available CRT graphic terminals. A comparison between that list and an updated version compiled from Computer Display Review,[2] and Modern Data Systems,[3] is given in Table I.

The number of suppliers has more than doubled in the past five years. During the past five years, several companies, such as Adage, IBM, and IDI, have offered upgraded versions of earlier systems. Probably the most widely used graphic terminals over the past five years were the IBM 2250 Series units. Originally introduced in Spring 1964 with the IBM 360 Series comupter, three additional versions were subsequently offered.

Two companies (Stromberg Carlson and Philco-Ford) have essentially withdrawn from the commercial field. Several companies do not appear in either list, because either they introduced and then withdrew products in the intervening years, or they introduced products and were then merged into another company. For example, Corning Data Systems and Graphics Display Ltd (England) both introduced low cost graphic terminals several years ago and then either formally, or informally, withdrew them from the market a year or so later. Computer Displays, Inc. introduced the first low cost graphic terminal (using a storage tube) about four years ago, and then merged into Adage about a year ago, losing its corporate identity.

I will not be surprised if there are other changes by the time this is published in May 1972 . . . new products and suppliers, mergers, or product withdrawals. For example, plasma panels and liquid crystal panels with associated displays are just now becoming commercially available from Owens-Illinois and Optel, respectively.

I estimate that the companies listed have each spent in the range of $250,000 to $3,000,000 to bring these commercial products into the market place. Perhaps, then, some $50,000,000 has been invested in these terminals, whose current installed value is about equal to that investment. Certainly, graphic terminal business is not a "get-rich-quick" scheme!

TERMINAL CONFIGURATIONS

Intelligent terminal

Five years ago, most terminals consisted of a display generator (with digital logic and some analog function generators) and a refreshed CRT. Only one system used a storage tube (the BBN Teleputer System), and only two systems included their own computers (DEC

TABLE I—Graphics Terminal Manufacturers
Available Configurations Now & Then

Company	Supplier	Intelligent Terminal	Storage Tube	Low Cost Graphic	Digital TV	Scan-Conv. TV	Unlimited Graphics Buffered	Unlimited Graphics Unbuffered
Adage	—/2	—/2	—/2	—/2	—	—	—	—
AEG-Telefunken (Germany)	—/2	—	—	—	—	—	—/2	—
Bunker-Ramo	—/2	—/2	—	—	—	—	—	—
Computek	—/2	—	—/2	—/2	—/2	—	—	—
Conograph	—/2	—/2	—/2	—/2	—	—	—	—
Control Data Corporation (CDC)	7/2	—/2	—	—	—	—	7/2	—
Data Disc	—/2	—	—	—/2	—/2	—	—	—
Digital Equipment Corporation (DEC)	7/2	7/2	—/2	—/2	—	—	—	7/2
Evans & Sutherland	—/2	—	—	—	—	—	—	—/2
Ferranti Ltd	7/2	—/2	—	—	—	—	—	7/2
Fujitsu	—/2	—	—	—	—	—	—	—/2
Hazeltine	—/2	—	—	—	—/2	—	—	—
Honeywell	—/2	—	—	—	—	—	—/2	—
Imlac	—/2	—/2	—	—/2	—	—	—	—
Information Displays, Inc. (IDI)	7/2	—/2	—	—/2	—	—	7/—	7/2
Information International (III)	7/2	—	—	—	—	—	—	7/2
International Business Machines (IBM)	7/2	—/2	—	—	—	—	7/2	—
International Computer (ICL)	—/2	—	—	—	—	—	—	—/2
International Tel & Tel (ITT)	7/2	—	—	—	—	—	—	7/2
Lundy	—/2	—	—	—	—	—	—	—/2
Marconi (England)	—/2	—	—	—	—	—	—	—/2
Monitor Systems	—/2	—	—	—	—/2	—	—/2	—/2
Philco-Ford	7/—	—	—	—	—	—	—	7/—
Princeton Electronics	—/2	—	—	—/2	—	—/2	—	—
Sanders Associates	7/2	—/2	—	—	—	—	7/2	—
SINTRA (France)	—/2	—	—	—	—	—	—	—/2
Standard Radio (Sweden)	—/2	—	—	—	—	—	—	—/2
Stromberg Carlson	7/—	—	—	—	—	—	7/—	—
Systems Engineering (SEL)	7/2	—/2	—	—	—	—	—	7/2
Systems Concepts	—/2	—/2	—	—	—	—	—	—/2
Tasker	7/2	—	—	—	—	—	7/2	7/2
Tektronix	—/2	—	—/2	—/2	—	—	—	—
Toshiba	—/2	—	—	—	—	—	—	—/2
Vector General	—/2	—	—	—	—	—	—/2	—/2
UNIVAC	7/2	—	—	—	—	—	7/2	7/2
Xerox Data Systems (XDS)	7/2	—	—	—	—	—	7/2	7/2

KEY
— No product.
7/2←Supplier in 1971/72.
└——Supplier in 1966/67.

and Bunker-Ramo). The other units used either non-programmable mass memories (such as core or drum) to refresh the display, or were refreshed from the core of the host computer.

In the past five years, the spectrum of configurations has significantly increased. Because of the sharp break in commercially available minicomputer prices, many more intelligent terminals[4] are now offered. A comment from the 1966 Computer Display Review[5] emphasizes the minicomputer price decline.

> "In fact, the DEC 338 has a general-purpose PDP-8 satellite computer which operates independently of the display controller. While the DEC display may seem expensive, the PDP-8 alone is worth $18,000."

Versions of the PDP-8 are now available for less than *one-third* of the 1966 price. Software supported intelligent terminals (which include their own commercial mini or midi GP computers) are now offered by Adage, Bunker-Ramo, CDC, DEC, IDI, IBM, Sanders and SEL. Conograph, Imlac and System Concepts furnish software supported intelligent terminals which use their own designed minicomputers.

Almost all other commercial graphic terminal suppliers are prepared to, or have interfaced their units to a variety of mini or other large scale host computers.

The 1966 Computer Display Review[5] could comment quite legitimately that:

> "There are presently no generally accepted standards or methods for evaluating line-drawing equipment."

In an effort to remedy the situation, the Computer Display Review developed a series of quantitative measures for refreshed displays, based on the manufacturers data. Figure 1 shows the range of price and performance for the displays included in the 1966 Review, compared to the 1971 Review. Note that although the data content characteristics have not changed significantly (the range of flicker-free points, lines, characters and frames), the minimum cost per function has in general been greatly reduced.

Low cost graphics terminals

Storage tubes have introduced one of the major changes in terminal configurations. Until about four years ago, essentially all graphic terminals used refreshed CRT's, with tube sizes ranging from 16″ round to 23″ round . . . resulting in usable display areas of about 10″ × 10″ up to about 14″ × 14″. After Tektronix introduced the Model 611 X-Y Storage Tube Unit with a 6″ × 8″ usable area, several companies including Computer Displays (now Adage), Computek, Tektronix, DEC, and Conograph, began to market interactive terminals based on the Model 611. These storage tube terminals marked the beginning of low cost CRT graphics . . . originally introduced in the $12,000 to $15,000 range, the units are now selling for about $8000. In late October 1971, Tektronix introduced a limited graphic storage terminal selling for less than $4000.

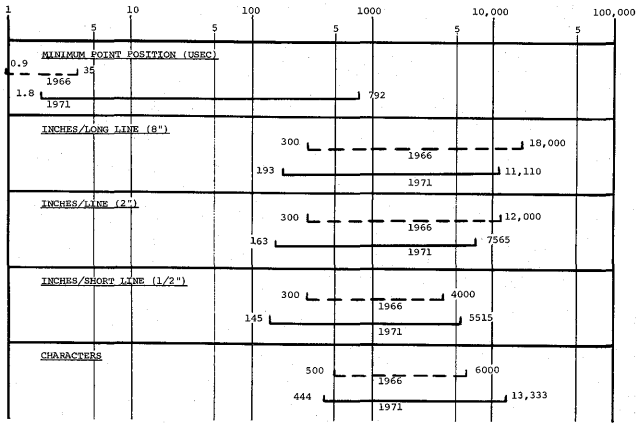


Figure 1a—Flicker free graphic functions

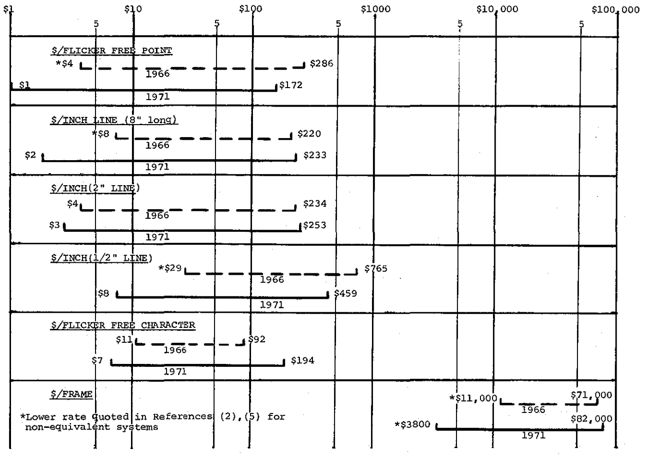


Figure 1b—Cost per graphic function

Within the past two years, several other low cost terminals (under $10,000), using either refreshed displays or some form of TV (either scan conversion or digital TV), have been introduced. Included are refresh displays from Imlac and IDI, scan converter displays from Princeton Electronics, and digital TV displays from Data Disc.

Generally, these low cost units are available with varying levels of software support.

Typically, the low cost graphics terminals involve some compromise in terminal performance . . . such as small picture area, low contrast, restricted dynamic motion, poorer picture quality (lower resolution and some line drawing limitations), and no gray levels. However, for many applications, these are acceptable compromises.

Long persistence phosphors

In order to increase the flicker-free data content of refreshed displays, many terminals now use long-persistence phosphor CRT's. Until about three years ago, the only satisfactory long persistence phosphor was the P7 . . . a combination P19 phosphor for persistence and P4 for fast response (in order to use a light pen). This phosphor couples reasonable burn resistance with satisfactory performance in the 30 frame per second refresh range.

TABLE II—3-D Picture Manipulation
Software vs. Hardware Comparison

(Adapted from Reference 30)

	SOFTWARE	HARDWARE	
ITEM		ANALOG	DIGITAL
1. Number of lines which can be rotated flicker-free.	Independent of rotation method. Depends only upon display techniques.		May bef unction of picture composition if average line drawing time less than matrix multiply time. Line content then determined by matrix multiply time.
2. Time to calculate constant data. Assumes approximately 600 machine cycles per calculation.	Independent of rotation method. Approximately 0.5 millisecond per new angular position.		
3. Time to calculate rotated point (line). (For Software, assumes approximately 170 machine cycles per calculation.)	130 USEC	1–6 USEC, due to transformation array settling time.	5–10 USEC
4. Resident program size.	Approx. 650 decimal words	Approximately 300 decimal words.	
5. Display list buffer size (including 3D and 2D display files), where *W* is number of words required to store 3D display file.	2.3W	W	W
6. Number of lines that can be *smoothly* dynamically rotated (without apparent jump from frame to frame).			
a. Maximum rate (180°/sec) beyond which eye gives "strobe" effect at 30 cps refresh, P31 phosphor.	250 lines	Limited by vector drawing time: number of vectors drawn @ 30 f/s.	
b. At 1°/second	2000 lines		
c. At 2°/second	1000 lines		
d. At 4°/second	500 lines		
7. Perspective	Yes, incl. in routines	No, can be simulated by Z dependent intensity modulation for depth cueing	Yes
8. Hidden Line	Yes, special cases processed in real-time.	No, requires software.	No, requires software.
9. Delta cost (approximate)	$5000 Assumes additional 4K memory increment required. (3D program 650 words; 2D display File #1 1675 words; 2D display File #2 1675 words.) However this increment can be used in other programs as well.	$15,000 $40,000	$70,000

TABLE III—Cost per Console Hour
(From Reference 7)

TYPE	CAPABILITY	INPUT	OUTPUT	EXAMPLE	COST PER HOUR
I Non-graphic teletype	alphanumerics only	keyboard	keyboard	ASR 33	$10
II Non-graphic refresher CRT	alphanumerics only	keyboard	display	IBM 2260	$15
III Graphic Storage Tube	alphanumerics and vector generation	keyboard	display	ARDS and Computek without tablet	$20
IV Graphic Storage Tube	alphanumerics & vector I/O	keyboard & tablet	display	ARDS and Computek with tablet	$30
V Graphic Refresher CRT	alphanumerics & vector I/O	keyboard, light pen, & function buttons	display	IBM 2250 CDC 274	$80 to $150

Recently, doped P39 phosphors, and the Ferranti L4 phosphor, have offered the same burn resistance with acceptable performance in the 16–25 frame per second refresh range.

Hardware function generators

Hardware vs. software trade-offs were continuously modified over the past five years. Most early systems included hardware line and character generation, some included circle generators, but picture manipulation and curve generation were done in software.

While this is still the predominant situation, some terminals including those manufactured by Adage, Conograph, Evans & Sutherland, Lundy, Sanders, and Vector General, offer 2-D and 3-D rotation hardware. Others, including those offered by Conograph, Lundy, and Sanders, offer some form of arbitrary curve generation hardware. Like most trade-offs, choosing hardware vs. software for these functions involve a clear understanding of the application in order to decide if the additional cost is warranted. Factors involved in such a trade-off study are illustrated in Table II.

Operator input devices

Over the past five years, the light pen and keyboard have persisted as the predominant operator input devices for graphic terminals. Joysticks and trackballs are used occasionally, and there has been continuing, although not large, interest in the SRI MOUSE.[6]

Rapidly assuming a major role as an operator input device is the Graphic Tablet. Until the advent of the storage tube display, the Graphic Tablet was viewed simply as direct competition to the light pen. Early versions, such as the Rand Tablet (supplied by BBN), were relatively expensive, (about $10,000 to $15,000), but there evolved a number of devoted users. Sylvania entered the market with an analog version, the price level came down somewhat (about $7000), but the lower priced light pen (about $1500) continued to dominate.

However, the light pen could not be used with storage tube systems, and much attention became directed to the development of a lower cost graphic tablet. Currently, at least two, under $3000, units are available; one from Science Accessories (the Graf Pen, using an acoustic principle) and the other from Computek (using a resistance technique). Undoubtedly others will be marketed.

Color displays

Five years ago, color displays could be most readily obtained with TV techniques, using the commercial, color mask tube. Although there were some isolated usage of the color mask tube in random (non-TV) systems, the systems were costly, and relatively difficult to keep satisfactorily aligned. TV was not widely used for Computer Graphics.

Several years ago, a new color tube, the Penetron, was introduced by several tube manufacturers, including Thomas, Sylvania and GE. The Penetron uses a dual phosphor, and color changes (over the range from red, through orange, to green) are obtained by switching the anode potential, usually over a range from 6000 to 12,000 volts. Switching times are currently in the order of 150 USEC/color, and the tube seems best used in a non-synchronous field sequential mode. Penetron systems offer essentially the same resolution as convention monochromatic random positioned systems (as compared to the lower resolution of commercial TV). At least one manufacturer (IDI) now offers the Penetron as an optional output display for both its low cost terminal (IDIgraf) and its higher cost intelligent terminal (IDIIOM), at a cost premium of about $8000 per display.

Deflection systems

Five years ago, most displays were magnetically deflected. In the terminals featuring fast hardware character generation (in the order of 10 USEC/character, or faster), the display usually included a second high speed deflection channel, either magnetic or electrostatic.

Currently, however, because of better deflection yoke design and improved transistor drivers, newer terminals, such as those supplied by IDI and Sanders, feature a single, wide bandwidth, magnetic deflection channel, capable of full screen deflection in 10 USEC, and capable of responding to characters written in about 3 USEC.

Improved tube and transistor design have also revived interest in electrostatically deflected displays. Storage tube systems use electrostatic deflection, but because of storage requirements, the writing speeds are relatively low. However, a new series of electrostatic, solid state, X-Y displays offered by Hewlett-Packard, feature fast deflection (1 USEC full screen), wide video bandwidth (5 MHz), good spot size (20 MIL), relatively large screen (up to 19″ rectangular), and low price (about $3000).

A terminal manufacturer can now also buy "off-the-shelf" magnetically deflected X-Y displays from suppliers such as Kratos and Optomation. Five years ago, Fairchild and ITT offered similar units, but they no longer market a commercial product.

APPLICATIONS

Five years ago, commercial usage of graphic terminals was limited almost exclusively to Computer Aided Design and Simulation. Many other applications were being investigated, but each investigator was essentially a pioneer. Except for the software supplied by a computer manufacturer to support his terminal (such as CDC and IBM), each user had to "start from scratch."

Today, the situation is considerably improved (although there is much more that can be done). Most intelligent terminal suppliers furnish some graphic software, including graphic subroutines, operating systems and higher level languages. Some offer complete application packages for free-standing versions of their systems (such as the IDI Automatic Drafting System, IDADS). Others (such as CDC and IDI) offer emulator packages that permit their terminals to appear like the IBM Series 2250 displays, and hence are capable of utilizing IBM, or IBM user, developed software. A number of systems organizations, such as Applicon and Computer Vision, also offer turnkey graphic terminal-based systems. These systems permit the user to use computer graphics for making PC boards and IC masks, without any further software investment. In fact, it appears now that most IC manufacturers are using terminal-based computer graphics.

Computer-Aided Design (CAD) remains a major application area, although reported usage is still concentrated in the Aerospace and Automotive industries. However, there appears to be increasing use in Architecture, Shipbuilding, and Civil Engineering.

Over the past five years, the use of graphic terminals in Utility Control has been accelerating. I estimate that about 10 percent of all investor owned utilities are now using or are planning to install graphic terminals for this purpose.

Enough commercial experience has been gained over the past few years to allow meaningful cost justifications to be prepared. The results of one survey giving console costs per hour billed to users for various types of consoles[7] are given in Table III. Five years ago, much justification for computer graphics was based on faith!

PUBLICATIONS, COURSES AND SEMINARS

Certainly, one measure of growth (or at least interest) in a field is the amount published, or the number of related discussions. In the past five years, several engineering level display and computer graphics texts, relating to computer displays, have been published[8,9,10,11] and numerous national and international meetings have been held. References 12–16 list several representative meetings. Several universities, including the University of California, the University of Michigan, Stanford, the University of Wisconsin, and Brooklyn Polytechnic, have sponsored short courses oriented to displays and computer graphics. ACM organized a special interest group for graphics (SIGGRAPH), and the Society for Information Display continues to flourish. Annual or periodic graphic terminal equipment surveys have become common like those published by Keydata* Corporation,[2,5] Modern Data Systems,[3] Auerbach Corporation,[17] Data Product News,[18] Computer Design,[19] and Computer Decisions.[20] Even the American Management Association,[21] the Harvard Business Review,[22] Scientific American,[23] Newsweek,[24] and the Jewish Museum[25] have

* In late 1971, Keydata Corporation sold its publishing business to GML Corporation.

taken note of computer graphics. Computer graphics terminals were featured on several national TV shows, like the David Frost Show and San Francisco Airport.

A few of the Seminars were concerned with "breast beating". It became increasingly popular in 1969 and 1970 to ask the question, "Why hasn't computer graphics lived up to its initial promise . . . a terminal in every home and office?"[26,27] Early predictions of a $200,000,000 market by 1970[28] were not being fulfilled. As a participant in many of these sessions, I felt that the question was being "begged". Some of the applications predicted for graphic terminals were being effectively handled by A/N CRT terminals (of which there are now estimated to be some 75,000 units installed).

The growth in other applications depended on a consolidation and analysis of results from the previous year's efforts. Still others couldn't be exploited until appropriate software, or less expensive hardware became available. And, 1970 was a miserable business year, anyway!

During these sessions, my position was, and continues to be, that although some early predictions were overly optimistic, conditions now exist for attractive growth. A number of market surveys, and projections have been published in the last five years . . . but the future is the province of another speaker in this session. Some review of the past five years might provide a useful bridge, however. For commercial applications, the consensus is that there are currently about 1200 high cost graphic terminals and about 700 low cost graphic terminals installed.[29] Five years ago, there were probably (my guesstimate) about 300 high cost graphic terminals installed. There were no low cost graphic terminals.

WHAT DIDN'T QUITE MAKE IT

As shown in Table I, almost all suppliers from five years ago are still offering commercial equipment. Several products and concepts which seemed promising during the period didn't quite make it though. For example, about four years ago, a British company, Graphic Displays Ltd, had an interesting idea for a low cost graphic terminal. The ETOM 2000 coupled an inexpensive drum memory to a long persistent phosphor display. Operator input was achieved with an X-Y mechanical table arrangement. Apparently technical problems and limited customer acceptance scuttled the project.

Corning Data Systems exploited a photochromic storage technique in their Corning 904 terminal. For about $20,000, the customer was offered a storage display with hard copy output, and extensive software support. But Corning couldn't find a large enough market and withdrew the product. All was not lost, however, because they were able to sell the software package to Tektronix.

Ported CRT's seemed to be a promising techn'que five years ago. However, the added cost and complexity limited the use to selected military applications, and there is little current commercial interest in the configuration.

Many practitioners expected (or at least hoped) that there would be a universal higher level graphics language by now . . . but that didn't quite make it, either!

SUMMARY

It was an exciting five years!

New suppliers, new products, and new applications surfaced during this period. Because of lower cost terminals and turnkey software/hardware systems, the use of graphic terminals began to spread beyond the Fortune 500 . . . beyond the Aerospace and Automotive Industries.

Generally, terminal performance was maintained, while prices were lowered. This was a reasonable trend since most applications were not hardware limited.

Of necessity, a survey paper like this tends to be superficial. For every example cited, several more may exist. But the purpose has been to give the sense of movement over the past five years, perhaps at the expense of some detail.

Usually, this kind of paper ends with a forecast . . . a prediction of things to come. Fortunately (since predictions have a habit of coming back to haunt), the seer's mantle has been placed firmly on another speaker's shoulders!

REFERENCES

1 C MACHOVER
Graphic CRT terminals—Characteristics of commercially available equipment
AFIPS CONFERENCE PROCEEDINGS Vol 31 1967 Fall Joint Computer Conference Thompson Book Company Washington D C

2 *Computer Display Review, 1971*
Keydata Corporation 108 Water Street Watertown Massachusetts

3 *Interactive CRT terminals, Part 1-full graphic CRT terminals & systems*
Modern Data June 1971 pp 44–55

4 C MACHOVER
The intelligent terminal
Paper presented at the University of Illinois "Pertinent Concepts in Computer Graphics" Conference 30 March–2 April 1969

5 *Computer Display Review, 1966*
Adams Associates Inc Bedford Massachusetts p V.214.0

6 C MACHOVER
Computer graphics in the United States
Computer Graphics Techniques & Applications Plenum Press New York NY 1969 pp 61–83

7 T J MOFFETT
The economics of computer graphics Part IV, cost effectiveness
Lecture notes from "Computer Graphics for Designers" University of Michigan Engineering Summer Conferences July 12–23 1971

8 H R LUXENBERG R M KUEHN
Display system engineering
McGraw-Hill New York NY 1968

9 S DAVIS
Computer data displays
Prentice-Hall Englewood Cliffs NJ 1969

10 S SHERR
Fundamentals of display system design
Wiley-Interscience New York NY 1970

11 M D PRINCE
Interactive graphics for computer-aided design
Addison-Wesley Publishing Company Reading Massachusetts 1971

12 *Computer Graphics '68 Seminar*
Brunel University Uxbridge England July 1968

13 *Computer Graphics '70 Seminar*
Brunel University Uxbridge England April 1970

14 *Recent advances in display media*
National Aeronautics & Space Administration, Washington DC A Symposium held in Cambridge Massachusetts 19–20 September 1967 Publication NASA SP-159

15 *Pertinent concepts in computer graphics*
Conference University of Illinois 30 March–2 April 1969

16 *DoD/Industry symposium on computer-aided design and computer-aided manufacturing/numerical control*
Rock Island Army Arsenal Davenport Iowa 14–17 October 1969

17 *Auerbach graphic processing reports*
Auerbach Info Inc Philadelphia Penna 1969

18 M MANDELL
A buying guide to visual display/CRT terminals
Data Product News October 1970 pp 1–7

19 J E BRYDEN
Visual displays for computers
Computer Design October 1971

20 *Graphic terminals—Man's window to interactive graphics*
Computer Decisions November 1971

21 *Computer-aided design engineering: the new technology for profit potentials in the seventies*
Session # 6311–60 American Management Assoc New York NY 5–7 October 1970

22 I M MILLER
Computer graphics for decision making
Harvard Business Review November–December 1969 pp 121–132

23 I E SUTHERLAND
Computer displays
Scientific American June 1970

24 *The artist and the computer*
Newsweek September 13 1971 pp 78–81

25 *Software Show*
The Jewish Museum New York NY 16 September–8 November 1970

26 *Interactive graphics . . . where is the market?*
Proceedings of Symposium Boston Mass 13 May 1969 Keydata Corporation Watertown Mass

27 *Interactive computer displays*
Data Systems News August–September 1970

28 D E WEISBERG
Computer-controlled graphical display: its applications and market
Computers & Automation May 1964 pp 29–31

29 *High growth plotted for computer graphics*
April 1971 pp 3–8 Samson Technology Trends Samson Science Corporation 245 Park Avenue New York NY

30 B WALDER C MACHOVER
Why ware . . . hardware vs. software rotation
Unpublished notes 29 October 1971

The future of computer graphics

by THOMAS G. HAGAN and ROBERT H. STOTZ

Adage Inc.
Boston, Massachusetts

Predicting the future is a hazardous enterprise—especially in high technology where seers tend to be too optimistic in the short run (3-5 years) and overcautious in the long run (10-20 years). In computer graphics, the forecasts started about 10 years ago, and so far almost all seem to have been too optimistic. But the pace now quickens, and the next 10 years' developments may at last outstrip the visions of 10 years ago. At least, so it seems now to us, and so we join the optimists, even though to date they've all been wrong.

The optimists (i.e., almost everyone in the field) have been disappointed because they failed to recognize at the outset the level of completeness necessary in a graphics system to do a useful job. They consequently either tried to work with incomplete systems, producing negligible results, or, in achieving completeness, encountered higher than expected costs for both development and subsequent production use. The realists, on the other hand, *knew* that achievement of successful on-line graphics applications would at first be a slow and costly process, and planned accordingly. They concentrated on applications areas where the high initial costs could be justified, and where the available hardware and software technology was sufficient for the job. They have achieved some significant successes. Based upon those achievements, we believe that computer graphics is entering a new phase which will be characterized by rapid growth.

We can summarize our conclusions for the future by saying that the technical feasibility of graphics applications has now been demonstrated in many areas, that economic feasibility has been established in a few areas, that costs will come down by a factor of 10 in 10 years, and that graphics systems will therefore proliferate, first in areas where the economics are already favorable, then into new ones as they become so.

Areas where growth can be expected on the basis of already proven economics include situation display, data analysis, and simulation. The economic turning point has just been reached for computer-aided engineering design (except for exotic cases which have been economic for several years) and for some forms of drafting. In the future lie still broader drafting applications, publishing industry applications for editing and page layout, architectural design, routine use for film and video tape animation and editing, management information systems, hospital information systems, and computer-aided instruction. We shall discuss each of these applications areas in turn, but first let us examine the technological prospects for computer graphics: the display hardware itself and the computer systems technology behind it.

DISPLAY HARDWARE TECHNOLOGY

Only five years ago, the random-access refreshed CRT was the only serious graphic display technology. Today, we have a plethora of new techniques. Interestingly, all of this new technology has been directed toward the image storage approach to graphic display. That is, the picture is stored in its two-dimensional form or in some one-to-one map of that form (e.g., video disk). Therefore, one cannot meaningfully discuss display technique without including the memory for retaining images.

In the next 10 years, it is reasonable to expect the bistable direct view storage tube (DVST), the video scan converter tube and the video disk each to mature and to find particular application in areas where its advantages will be realized. A long life for each approach can be expected, although, particularly for the DVST, new hardware will probably succeed present equipment.

One new direct view storage technique is the plasma panel, which appears to be nearing commercial application. The plasma panel performs functionally like the bistable DVST; that is, it retains images for direct view once it has been written. But it has some interesting advantages. It is fundamentally flat and reasonably transparent, which makes rear projection of static data easy. It should be more rugged than a CRT and it is digitally addressed, thereby eliminating all analog circuitry. But its most interesting feature may be its price. The cost of this panel has been projected by authoritative sources to be as low as a few hundred dollars for a panel with drivers. At any price below $1000, the plasma panel will certainly be a popular device.

Several other DVST techniques have been introduced, but we do not foresee major impact from any of them. Photochromic storage systems are slow and too expensive and, in fact, the one commercial entry utilizing a photochromic system has been withdrawn from the market. The cathodochromic direct view storage tube is viewed by reflected light, which is an advantage in a high ambient light environment. However, its slow erase property will limit its utility.

The random-access CRT does not have much competition today where dynamic performance is required. There will be a shift toward raster displays, however, possibly for even the most dynamic pictures. The flexibility of TV video format for remote transmission, large screen display, and image mixing, particularly with non-computer generated pictures, argues for this, as does system cost. In the past, the scan conversion cost was prohibitive, but as digital logic cost drops further and its speed improves, the cost will become more competitive.

The move toward video format will encourage the work being done on display of solid figures. However, applications for solid figure display (limited more by computer systems technology than display hardware) will probably continue to be limited, even in 10 years.

Quite likely, too, by the early 1980's, several new display techniques will be with us that we are not even aware of today. It is hard to project the impact of currently unknown techniques, but, as has been the case in the past five years, they will probably be directed more at cost reduction than at performance improvement. In fact, in the next decade, it may be that *all* the significant technological developments in display devices will result in cost reduction rather than in performance improvement.

COMPUTER SYSTEMS TECHNOLOGY

The factor that has limited the utility, and therefore the market growth, of interactive computer graphics systems has not been the display hardware technology (CRT's have worked fine for years) but the computer systems technology behind the display. By this we refer to the whole system for providing enough on-line processing power, sufficiently responsive, with enough memory to be effective, with adequate programming support, and with communications adequate to get the data to the user at the necessary speed, all at a low enough cost.

Interactive graphics systems are by definition conversational. In order to be truly interactive, they must provide responses within a few seconds for many or most of the actions taken by the user at the console. Cost for graphics systems has always seemed high to users, and attention has often been focused on the cost of the display terminal itself as an obvious target for reduction. But compared to keyboard-only conversational systems, conversational graphics entails manipulation of relatively large data files, with consequently much longer messages between terminal and mass memory. Concentration on cost of the display device often has led to overlooking the cost of the processor hardware and the mass memory necessary to support the terminal device with the levels of performance needed for a conversational graphics environment. In the past, some systems disappointed users because of the large amount of core memory needed in the system to support each terminal, or the high cost of the communications facilities necessary to provide the link to the CRT, or the lack of enough fast mass memory to support a reasonably complete application. Proliferation of graphics systems, therefore, depends not only upon reducing cost of the display itself, but also upon reducing the cost of the processor hardware and the mass memory needed to support the display. Numerous users have concluded that each graphics console requires in the range of 10 million to one hundred million bits of mass memory with access time well below 1 second. Communications costs are significant whenever this mass memory and the processing hardware are remote from the console. Even though present digital communications costs are expected to be reduced dramatically, we believe that many or most graphics systems will incorporate local processing power, and some form of local mass memory. The communications burden then reduces to one of occasionally shifting data from individual consoles or clusters of consoles through communications lines to a central repository accessible to many users.

The need for sufficient processing power, core memory, and mass memory behind the tube is the reason that useful interactive graphics systems today cost $100,000, and up, per console. We believe that 10 years will see a 10:1 reduction in these costs, permitting complete, useful systems with the equivalent of today's functional performance for $10,000. Cost per console hour will therefore drop from today's $25.00 to $50.00 range to about $2.50, permitting significant growth of current applications, and penetration of computer graphics techniques into many areas currently untouchable. Here are our thoughts on prospects for a number of different present and future applications.

APPLICATIONS

Many people assume that Interactive Graphics = Computer Aided Design (CAD). It is true that CAD is an important applications area for computer graphics and the one that most easily captures the imagination, but there are many other application areas, including some that have achieved a higher degree of maturity (i.e., proven economic effectiveness) than CAD. We shall discuss a series of application areas in turn, roughly in the order of their currently demonstrated effectiveness. (We are specifically not discussing them in the order of present or potential market sizes, which cover a wide range, although clearly areas where economic effectiveness is established soonest generally achieve earliest penetration of the available potential market.)

Situation display

About $100 million has already been committed by the FAA for computer graphics systems for air traffic control situation displays. The next 10 years should see computer graphics firmly established for real-time situation displays in many areas. Other forms of traffic will be brought under control of systems which will include computer graphics for overall situation display: rail, transit, harbors, shipping lanes. Computers will be inserted between the antennae and CRT's of almost all radar systems, converting them into computer graphics systems capable of much more comprehensive situation display than current "simple" radars. Thus, specialized computer graphics systems will eventually be found aboard all airliners and all ships as part of the navigation and piloting systems—if not by 1982, then by 2001, just as we were informed by Stanley Kubrick's film.

The few computer graphics systems now installed for monitoring and control of chemical processes and utility distribution systems presage what we believe will be widespread use of computer graphics for situation display in these areas, possibly replacing completely the walls full of panel meters and recorders that have come to typify the industrial process control room.

As discrete manufacturing operations become more automatic, situation displays will begin to be used for monitoring material flow and inventory levels. In other words, for supervisory control, somewhat analogous to that practiced in continuous flow chemical manufacturing processes.

Computer graphics systems are already being used as situation displays for monitoring tests and experiments conducted with the use of expensive test facilities, and frequently for assisting with control of the facility. Aircraft flight test, wind tunnel test, and Van de Graaf generator experiments are all today being conducted with computer graphics systems used for real-time display and control. Achievement of even a small increment of increased utilization (or enhanced safety) of a very expensive test facility can justify a computer graphics situation display system at today's costs. As costs come down, such display systems will proliferate, finding use in conjunction with larger numbers of gradually less expensive test facilities.

Data analysis

Any operation involving the collection of large quantities of data for subsequent human interpretation constitutes a potential application area for computer graphics. Today's successful applications (in the military, electronic intelligence data analysis; in industry, oil exploration data analysis) can be expected to continue growing and should lead to applications in similar areas: semi-automatic map-making (photogrammetry), weather data reporting and forecasting, and other forms of mineralogical survey data analysis besides oil exploration. Increased use of automatic sensors to collect data should lead to increased use of computer graphics as an aid to analysis of that data. In addition, some data sets built up by keyboard source data entry (census data, for example) have become large enough to be candidates for analysis via computer graphics.

Simulation

This is an area that has used computers in an on-line, interactive mode right from the start, and usually with some form of graphic output—strip charts or X-Y plots. Analog computers were used almost exclusively at first, then hybrids took over, and now much simulation is done on purely digital machines. Simulation models have grown more and more complex so that systems behavior can be studied *in toto* rather than piecemeal. Flexible dynamic display has become more and more necessary, either to achieve "picture-out-the-window" verisimilitude in the case of vehicular simulations, or to achieve an over-all "situation display" for the simulation model—with the same general requirements (plus flexibility) required for the "real world" situation displays discussed above.

To date, engineering simulation has been highly mathematical and quantitative, even where the object was to get the "feel" of sensitivity to parameter variations. The display flexibility of computer graphics will permit a more qualitative kind of simulation, still based upon an underlying mathematical model, but where there is less a "numerical" and more a "graphical" sense of the model. In short, simulation will be used for what we think of as styling as well as for what we think of as engineering design. The distinction between the two, already blurred, may gradually disappear. In the future, dynamic aspects will be "styled," as static aspects now are.

Computer aided design

Meanwhile, there is CAD. After a long slow start, great strides are being made and numerous on-line applications are felt by their users to be cost-effective today. However, the dream of an integrated design system which takes a large design task from concept to final production drawing (or microfilm or magnetic tape) will be realized in only a few areas even in 10 years. Clearly one such area is the layout and design of electronic circuits. In 10 years, virtually all circuits will be designed with the aid of the computer. Circuit and logic designs will be synthesized and tested by computer, and test specifications for completed circuits will be an automatic by-product. Almost all of this kind of design work will be done with on-line interaction via graphic terminals.

One consequence, in this and other areas, will be a change in the economics of short-run production. By cutting design costs, CAD will make the production of "custom" circuits more practical.*

Interactive graphics are being used successfully today in several dozen places for performing various kinds of engineering analyses that form a part of the design process. Success has come quickest where a pre-existing and already proven applications package developed for batch operation has been adapted to be made available on a conversational basis via interactive graphics. The usual main result is a dramatic shortening of the cycle time for the part of the design job converted to graphics—a job involving many iterations that used to take six weeks under batch being finished in one two-hour console session, etc. There is also often the feeling (hard to prove) that a better design job was done. The high cost of graphics systems and applications programming today restricts these applications to a few critical points in the design process. As costs come down and confidence increases, use is spreading to less critical design jobs: from today's computation-intensive jobs (stress analysis, heat flow, vibration analysis) to geometry-intensive jobs (complex layouts, clearance analysis, pipe routing, numerically-controlled machine tool tape generation and verification) and finally to drafting/clerical intensive jobs (detailed drafting).

Some forms of detailed drafting are already cost-effective using computer graphics, in particular the preparation, editing, revision and file maintenance for drawings made up from a set of standard symbols. (Circuit diagram, logic diagrams, flow charts, block diagrams, etc.) Especially where revision is frequent and fast turn-around is important, as in the preparation and revision of system maintenance drawings, computer graphics is now ready to compete with traditional drafting and technical illustration methods.

The design function is a social activity. Design tasks are usually undertaken by groups of people, with the overall design job in one way or another divided among members of the group. It is easy to think of CAD as a means for helping an individual designer with his task, and to overlook the implications of CAD with regard to communications within the design group. In thinking about the long run, it is important to visualize teams of designers using on-line interactive devices to communicate with a data base containing both the design constraints they are working with (specifications, costs, catalogs of available parts) and the results to date of their joint effort. Much of their communication with one another will be via the design itself as it exists in storage. What will happen when (and if) 10, 100, 1000 designers are interconnected on line? The airlines reservations systems constitute the closest current model to such a situation—and the detailed total reservation schedules the thousands of clerks at CRT terminals are constantly building up is a kind of "design"—but the constraints they work with are few and fairly simple.

Architecture

We think the introduction of computer graphics here will be relatively slow because of the diffuse nature of

* Similarly, wherever the high cost of processing information "by hand," for either the design or the manufacturing portion of the production process, has resulted in a large cost differential between "custom" and "mass" production, CAD, with computer graphics playing a significant role, may redress the balance, and much of our industrial output may return to the "one-of-a-kind" production typical of pre-industrial times. If so, computers will have (in this respect at least) an effect upon the quality of our lives quite opposite from that expected by those who see them only as a threatening source of increased standardization and regimentation.

the business structure, with many small independent firms, most without the resources for investment in hardware and development of software. City and regional planning may be an exception if government money is used to fund the necessary development. Emphasis upon the creation of presentation drawings is misplaced. The cost of renderings and models for client presentations represents a small part of the total architectural cost, the bulk consisting, as in other design activities, of the creation of working drawings. Therefore architecture may benefit mostly from progress made elsewhere with generalized drafting: the creation of dimensional drawings with materials lists.

Publishing

Computer graphics has promise as a publishing tool, and we expect substantial progress in this area in the next 10 years. Preparation and editing of text and graphics source material, and page layout, are the functions that graphics systems can help with. Automatic typesetting has gone as far as it can go without the front end of the publishing process being automated. Cost reduction and shortening of the cycle time are again the primary benefits. By 1982, it will no longer be necessary to submit SJCC papers "in final form suitable for printing" seven months prior to publication of the Proceedings, as it now is. But, in order for graphics to see widespread use for this function, it is clear that console time must first get most of the way down to that $2.50 per hour.

Film and video tape production

There is an obvious role for computer graphics for producing and editing visuals (as they are called in the trade) for television and motion pictures. Entertainment programs, commercials and educational programs have all been produced on a limited but already commercial scale. The first full length feature movie shot and edited solely on video tape, then transferred to standard 35 mm color film, has been produced and distributed. We expect to see increased use of computer graphics as combined animation stand and editing console, both for creating "computer generated pictures"—i.e., computer animation—and also for editing live-action film or VTR footage.

Computer aided instruction

The current high-priced applications such as trainers for airplane pilots and astronauts are of course already economic, but normal classroom use of computer graphics as a standard teaching tool, which is what most people mean when they talk about CAI, clearly requires system costs in the range of $2.50 per hour and down, and therefore we do not believe that the next 10 years will see massive proliferation. On the other hand, small computer graphics systems will become standard lecture/demonstration tools in many areas of education, both for "live" use and for creating simple low- or no-budget teaching films or tapes—i.e., canning lectures.

Management information systems

Of course every organization has *some* kind of management information system—partly verbal, partly a manual paper system, and in most large organizations these days, at least partly computerized. But the computer part is batch, not on-line, so it isn't what people mean who talk about MIS. So far it's been mostly talk, with not much in the way of results. Maybe because on-line MIS isn't needed—top management really doesn't care what goes on minute by minute, and counts itself lucky to get reliable reports weekly or monthly. But down in the infra-structure, there *is* a need to know minute by minute about material location, tool availability, stockroom levels, etc. If on-line computer systems begin to get applied for these functions, then MIS as commonly envisioned will be a by-product, and computer graphics will be a popular tool for analyzing the data contained in such systems.

Hospital information systems

Many people seem to believe that Hospital Information Systems, as distinct from other types of computerized management information systems, are needed, feasible, and about to happen. If so, there will be a role for graphics, since so much patient care data are graphical in nature. The need will be for low cost inquiry stations capable of displaying static charts, graphs, and possibly even line drawings abstracted from X-rays. These would be terminals connected into an overall integrated information system primarily concerned with record keeping.

On the other hand, there will be another use for computer graphics in the form of higher cost, more dynamic systems used for patient monitoring in operating rooms and intensive care wards. Use of these systems will not depend upon creation of an integrated HIS.

CONCLUSIONS

In interactive computer graphics, we are at the end of the beginning. About $50 million has been invested to develop the products now on the market. Users and government sponsors have invested an additional $20 to $40 million in software development to tell us what we now know can and can't be done working in front of a CRT. In keeping with the New Accounting, almost all of these development costs have been written off as losses by the various organizations that have made investments to get started in graphics. Now computer graphics has entered a new phase. Most of the systems currently installed (about $100 million worth of hardware, by our estimate) were ordered as interesting but risky and somewhat daring experiments—many of which failed; but in the last year, graphics systems began to be ordered to do specific jobs, with reasonable certainty that they would be effective in those jobs. Thus we believe that an important corner has been turned.

Two ingredients are necessary for the field of computer graphics to experience growth—the availability of systems complete enough in hardware and software to do a variety of useful jobs, and a pattern of decreasing costs to end users. Both of these ingredients are now present, and we therefore believe that the period of significant growth for computer graphics, long anticipated, has now begun, and will be sustained throughout the coming decade.

A versatile computer driven display system for the classroom

by J. W. WILLHIDE

Boston University
Boston, Massachusetts

INTRODUCTION

The last two decades have seen education in the sciences, mathematics and engineering move from a practical base to the teaching of highly sophisticated abstract principles. The changes have been rapid and radical. Generally, they have affected subject matter only and for the most part this new material is being taught by traditional methods. The changes in subject matter are proper, but they must be accompanied by changes in pedagogy. The use of display-based, highly interactive computer simulations represents one approach toward building the required new pedagogy.

The educational system in this country is almost totally classroom based—one teacher interacting with a class of students on a regularly scheduled basis. This unanimity of approach implies that most administrators feel this is the most cost-effective approach to teaching *available today* that lies within their financial resources. In a similar fashion, one of the most cost-effective ways to implement a model-oriented pedagogy *today* is through the use of a computer-driven display system in the classroom. Other approaches to a model-oriented pedagogy such as individual student terminals supported by a CAI (Computer-Aided Instruction) system may at some point in the future challenge the classroom system on a cost-effective basis. However, it will be some time until such an approach will become the rule rather than the exception. One must bear in mind that even if CAI systems offered a cost-effective advantage over the classroom system today, many schools could not afford such a system because of the large initial investment and relatively long pay-back period.

At Boston University, the College of Engineering has implemented a versatile computer-driven classroom display system known as the AID System. The AID System is unique in its emphasis on real-time dynamic graphics, i.e., computer generated animation. AID allows the user to animate, in real-time, iconic representations of physical systems in a highly interactive environment. The user can change parameters and immediately observe the effect of the change on a CRT display. In addition to its real-time dynamic graphics capabilities, the system also makes use of the more conventional static information transmittal modes such as alphanumeric text, graphs and symbolic diagrams.

The AID System is structured around a PDP-9/AD-40 hybrid computer. The AD-40 analog computer is used to simulate the system under study or serve as a signal processor for experimentally derived data. Animation is produced by dynamically manipulating a set of geometric figures generated on the PDP-9 digital computer as functions of the time-varying signals sampled from the analog computer. These animation sequences can be accompanied by graphs which unfold in time synchronism with the animation. For cases where animation does not make sense, graphs of selected simulation variables can be displayed.

A high-level language, the AID Language, is under development to support the system. This language can be easily learned by a highly motivated sophomore engineering student. It is not intended, however, for the professional animator or casual user of the system, since some background in analog and digital computation is assumed. A film illustrating some of the attributes of the AID System and produced on the system itself using the AID Language will be shown.

ROLE OF THE SYSTEM

The system serves a dual role in the classroom. First, it can be used to assist the student in grasping difficult concepts through the use of iconic and graphical presentations. Second, since the system is computer-based, it can be used to illustrate the role of simulation and computation in engineering design. The above

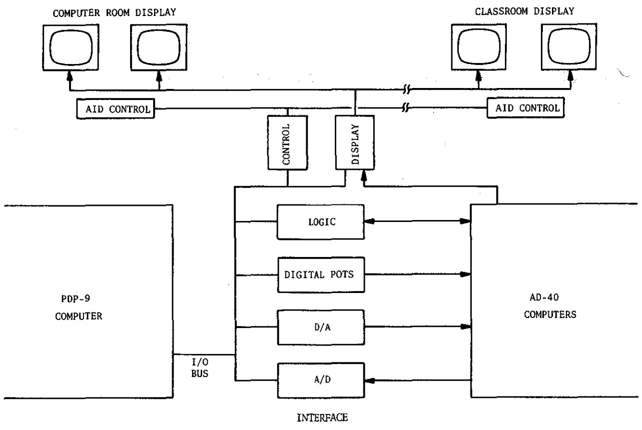


Figure 1—AID system

dichotomy is not always clear in many classroom situations. However, it does serve to focus attention on the two fundamentally different roles of the computer in the system—as an invisible "helper" and as a visible computer performing calculations and other tasks that a student thinks a computer should do.

The classroom-oriented display system can also serve several useful functions outside the classroom. First, since a duplicate display system is also located in the computer room, highly inquisitive or troubled students can interact with a preprogrammed simulation sequence on an individual basis and often answer their own questions. Meaningful interaction can be achieved with minimal programmed support because the student has been introduced to the simulation sequence in the classroom. A second function of the system outside the classroom is with student use of the system hardware and supporting software to add a new dimension to their project work. As an example, a group of sophomore-level students programmed, as an extracurricular activity, an animation of a pendulum swinging in an accelerating railroad car to accompany a project in their Dynamics course. The third function the system can perform outside the classroom is in conjunction with the laboratory exercises associated with our introductory analog computer course. In these exercises the students animate, in real-time, a pictorial representation of the physical system they are studying by signals generated in their analog computer simulation. The resulting iconic communication mode helps students bridge the gap between the typical "squiggly-line" output of the analog computer and the physical system which they are simulating. In addition, it has been found that the introduction of computer animation into an undergraduate course brings with it a fresh new breath of life—a level of excitement and motivation rarely seen today.

The system, with its dynamic systems simulation and graphics capabilities is ideal for generating demonstrations in a wide variety of undergraduate engineering disciplines. We hope to add interest to lectures while clarifying points which are difficult to perceive in terms of verbal descriptions or blackboard sketches. The demonstration examples will in some cases precede the lecture material to motivate and prepare the student for the abstract material. In other cases, the demonstrations will follow the presentation of a concept to validate and reinforce that concept. In both cases it is expected that the real-time interaction offered by the system will considerably heighten the sense of discovery and intellectual curiosity of the students in the classroom.

SYSTEM CONFIGURATION

The classroom display system at Boston University is depicted in Figure 1. From the figure it can be noted that the bulk of the system is a typical hybrid computer configuration consisting of a Digital Equipment Corp. PDP-9 digital computer, two Applied Dynamics AD-40 analog computers and a custom designed interface. Thus, the step from a hybrid computer to an interactive classroom display system is a relatively small one—consisting of adding a Control Interface with remote control units, a Display Interface and large screen CRT displays.

A hybrid computer was chosen to drive the system to gain the complementary attributes of its constituent analog and digital computers. The analog computer offers a convenient means to simulate the dynamic systems used in most classroom presentations. The digital computer is used to give the user a simple and yet effective interaction with the simulation through the use of a special control box located in the classroom. Through its graphics abilities, it also gives the system alphanumerics for giving instructions to the user and displaying parameter values, and line graphics for static and dynamic pictorial presentations.

Two independent large screen CRT displays placed side-by-side are used in the classroom. This provides the system with what the author believes is the visual channel capacity required for a flexible and effective computer-driven interactive classroom display system. For example, in demonstrating the response of an electric circuit to some excitation, one display would present the schematic diagram of the circuit complete with parameter values while the other display would present the excitation and circuit response. Once given a dual display system, it seems that almost all demonstrations require such a system for effective presentations. This may either be a corollary to Parkinson's Law, or, as the author interprets it, an indication that two independent displays are essential.

DISPLAY SUBSYSTEM

Of the two displays in the AID System, one is primarily used for the presentation of static or slowly changing images such as alphanumeric text, pictorial diagrams and graphs. This display, the left-hand one of the pair, is a large screen TV monitor. It is driven by a Tektronix 4501 Scan Converter, a device which accepts $x-y$ inputs and produces a video output signal. The scan converter is based on the use of a bistable storage CRT. Hence, this display functions as a stored type display. It need only be written once and it will retain the stored image until erased. When driven by a digital computer, the stored type display places considerably less burden on the computer than a refreshed type display. The stored type display is also convenient to have for many analog computer generated displays. For example, a simulation may be running in the repetitive operation mode with one value of a particular variable being generated during each run. A storage type display is required to present a plot of this variable versus time in the classroom.

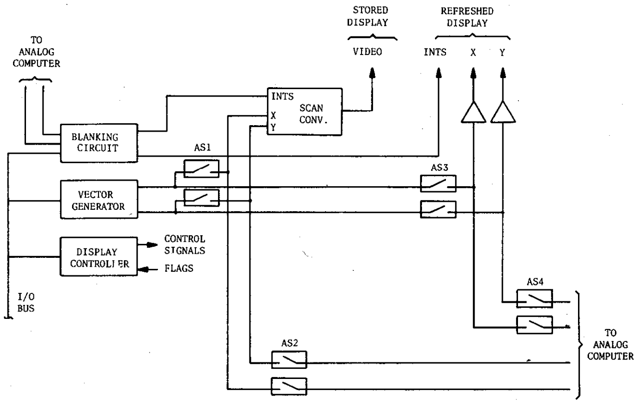


Figure 2—Functional diagram of display interface

The other display of the system is used primarily for the presentation of dynamic graphics such as animated iconic models (e.g., a pictorial spring-mass system which looks and behaves like a real spring-mass system) and curves generated by the analog computer in the repetitive operation mode. This display is a large screen $x-y$ CRT which functions as a refreshed display. It is driven directly by analog signals from either the analog computer or the digital computer display interface.

					Display Assignment	
Configuration	AS1	AS2	AS3	AS4	Stored	Refreshed
1	closed	open	closed	open	DIGITAL	DIGITAL
2	closed	open	open	closed	DIGITAL	ANALOG
3	open	closed	closed	open	ANALOG	DIGITAL
4	open	closed	open	closed	ANALOG	ANALOG

Figure 3—Source/display combinations

A functional diagram of the AID Display Interface is given in Figure 2. Analog switches which are under program control are used to set the system configuration to suit the particular problem being solved. Figure 3 presents the various combinations between driving sources and displays. When both displays are driven from the digital computer, the programmer selects the display he wishes to write through the use of an appropriate display command. Both displays receive the same $x-y$ inputs. However, only the one being written receives an intensify pulse. This allows the programmer to switch back and forth between the displays. When the displays are driven by the analog computer, display selection is achieved through trunkline patching.

CLASSROOM CONTROL UNIT

A remote control unit is used in the classroom to give real-time interactive control to the system user. A pictorial representation of this unit is given in Figure 4. It was designed to fill the dual objective of presenting to the user an extremely simple and yet versatile interface with the system while at the same time minimizing the hardware and software needed to support the unit. The AID Interactive Control Unit allows the user to:

(1) Set and change parameters.
(2) Set initial conditions.

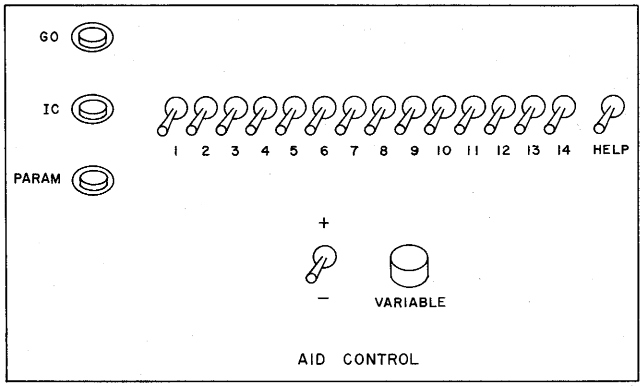


Figure 4—AID interactive control unit

(3) Choose a preprogrammed presentation mode, i.e., what variables will be displayed, what parameter will be variable, etc.

(4) Request HELP messages, e.g., switch assignments, presentation modes available and how to use them, etc.

A more detailed look at the classroom use of the AID Interactive Control Unit will be given in the following section.

All interactive parameter and initial condition values are entered into the system via a single potentiometer on the control unit. The analog signal from this potentiometer is read into the PDP-9 computer through a channel of the A/D converter in the hybrid computer interface. The digital computer then either sets the appropriate digital pot on the analog computer or changes a program variable. It also presents the classroom user with a "digital voltmeter" type display on one of the display units so that he can set the variable to the desired value. This visual feedback technique not only compensates for resistive voltage drop in the long lines to the computer room but also couples the class in with "the action" and visually indicates to the students the current value of the parameter being changed.

When a user initiated response is requested by depressing one of the three pushbuttons, a serial data stream containing information on the pushbutton depressed and states of the switches is transmitted over a single line to the Control Unit Interface in the computer room. When the complete message has been received, a program interrupt request to the PDP-9 computer is initiated. The computer responds to this interrupt and branches to an appropriate point in the AID program to carry out the requested response.

AN ILLUSTRATIVE EXAMPLE

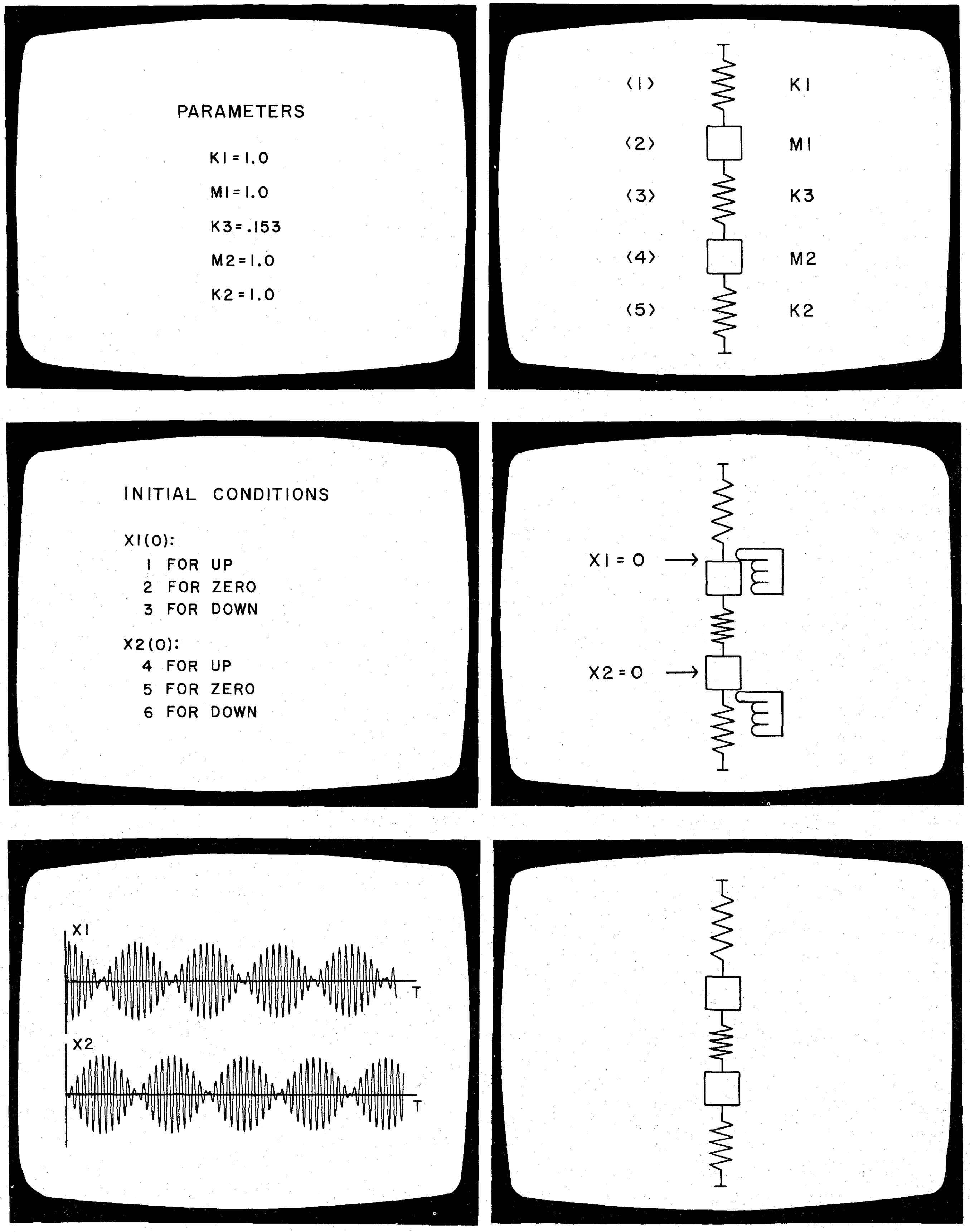
Several segments from a presentation developed on coupled oscillators will be used to illustrate the type of typical interactive classroom use which can be made of the system. When first initiated, the system presents a title, a "what to do" message and then awaits further commands from the classroom control unit. For the example we are considering, the left-hand display would present "MASS-SPRING SYSTEM . . . FOR NORMAL SEQUENCING HIT 'GO' AT THE END OF EACH STEP . . . OTHERWISE USE 'HELP' SWITCH IN CONJUNCTION WITH 'GO' FOR DIRECTIONS." In normal sequencing, the first depression of the GO button takes the system to the parameter mode. In this mode the user can change any of the parameter values from those nominally entered. He does this by setting the appropriate switch, hitting the PARAM pushbutton and then adjusting the potentiometer on the control unit for the desired value while watching the "digital voltmeter" on the display. This is illustrated by the top display pair of Figure 5. The user is in the process of changing K3 from the nominally entered value of 1.0 to a new value of 0.100. The numbers in the ⟨ ⟩'s are the switch assignments for the parameters. Other parameter changes are made by repeating the above sequence. The next depression of the GO button takes the system to the initial condition mode. For this particular example, the user does not set any values for initial conditions but simply chooses the perturbations to be made on the mass-spring system. When the IC pushbutton is depressed, the masses are gently nudged to their initial condition position by AID's "helpful little computer hands." The middle display pair of Figure 5 illustrates the initial condition mode—it can be thought of as a snapshot of the displays as the hands are in the process of moving the masses to their initial condition positions (down for M1 and zero for M2). Subsequent depressions of the GO button take the system through various run modes such as the one illustrated in the bottom display pair of Figure 5. For the case shown, an animation of the bobbing mass-spring system is on one display while the corresponding displacement of each mass from equilibrium position is plotted on the other display. Other run sequences plot various combinations of accelerations, velocities and energies.

The above example illustrated the user following normal sequencing by repeated depressions of the GO button. By using the switches in conjunction with the GO button, the user can break the normal sequence and branch to any particular desired segment of the presentation. Through use of the PARAM and IC pushbuttons, parameters and initial conditions can be changed at any time during a presentation.

COMPUTER ROOM CONSOLE

A special console is used by the programmer in developing AID programs. It features dual displays whose characteristics are identical to those in the classroom, an AID Interactive Control Unit, and a Computek GT50/10 graphic tablet. Thus, all phases of program development ranging from initial image generation to final debugging and checkout can be done at the console.

The console, in conjunction with the AID Language, will not only make the programming of AID presentations convenient and easy but will also provide a stimulating environment in which to work. We are de-



PARAMETERS

K1 = 1.0

M1 = 1.0

K3 = .153

M2 = 1.0

K2 = 1.0

INITIAL CONDITIONS

X1(0):
1 FOR UP
2 FOR ZERO
3 FOR DOWN

X2(0):
4 FOR UP
5 FOR ZERO
6 FOR DOWN

Figure 5—Illustrative classroom display sequence

pending on the latter attribute coupled with the drive for relevance and accomplishment on the part of today's students to supply a manpower pool to assist the instructional staff in developing AID presentations.

SOFTWARE SUPPORT

The AID Language under development to support the system is in an experimental phase and might best be described as a pseudo high-level language at this point in time. It has many of the appearances and attributes of a high-level language but has no compiler or interpreter. In its present form it is simply a collection of subroutines and symbol definitions that are used in conjunction with the PDP-9 assembly language programming system. This approach allows for changes to be quickly and easily implemented—an important consideration during the early experimental stages of developing a language. Its chief disadvantages are a slight awkwardness in the association of control variables with their commands—they must follow in a vertical string underneath the command—and the use of an octal rather than a decimal number system base. However, even in this experimental form the language is easy enough to use such that sophomore students can write program sequences.

The language provides commands for:

(1) Servicing the interactive control unit.
(2) Controlling the analog computer.
(3) Generating static and dynamic graphical displays.

It features an on-line mode which allows the program developer to perform the following functions interactively while sitting at the computer room console:

(1) Write his program in a conversational mode.
(2) Enter graphical images via a graphic tablet.
(3) Scale, move and synthesize graphical images.
(4) Generate, scale and move alphanumeric text displays.
(5) Tune up timing used in sequencing.

To illustrate the basic structure of the AID Language, an example AID program is given in Figure 6. This example shows how the top support, spring, and mass of the MASS-SPRING SYSTEM discussed earlier are generated. The analog computer is solving the differential equations governing the system in real-time. From these system variables, the analog computer generates the dynamic animation variables used to animate the static images stored in the PDP-9 computer. The dynamic animation variables are read into the digital computer on each pass through the program by the SAMPLE commands.

In order to avoid becoming prematurely display-bound in a system designed for real-time dynamic graphics it is essential that a continuous stream of data flow to the display. Compound images must not be constructed in memory and then displayed. They must be displayed as they are constructed. This philosophy is inherent in the instruction pair TRANSFER/CONTINUE. One point at a time is transferred from a source table to a specified destination—usually a display. During this transfer of the image from the source table, any of the AID manipulative commands placed between the TRANSFER and CONTINUE command operate on the image being transferred. All images which are to be dynamically displayed are stored centered around the origin of the coordinate system used for the displays. This system has its origin located in the center of the display area. Such an approach simplifies the definition and generation of the dynamic animation variables.

In the example of Figure 6 the image SPRING, which consists only of the zig-zag lines, was dynamically scaled in the x and y directions to convey the illusion of a spring being alternately stretched and then compressed. The invariant vertical segments of the spring were associated with the images TOP and MASS. Another, and even better approach, is to use the command ANIMATE in conjunction with two key frames stored in memory. In this case, the invariant vertical segments of the spring could be associated with the stored images of the spring rather than the mass and top support. The number of control variables would be reduced from three to two—one for translation and one to govern the amount of linear interpolation between the key frames specified by the ANIMATE command.

The header statement VIEW runs the program in real-time for direct viewing in the classroom or computer room. When making a film, this mode allows for previewing a sequence before it is actually recorded on film. Once a satisfactory sequence has been obtained, the statement CAMERA is substituted for VIEW and the system is ready for filming. The header statement CAMERA automatically slows the system timing down for filming at four frames per second and synchronizes the computer to the movie camera.

FUTURE PLANS

We view the AID System configuration of Figure 1 only as a starting point in our endeavor to couple the computer effectively into the classroom. For example,

```
LINE Ø/     NEW                  Initializes the system for a new run from t = 0

LINE 1/     VIEW                 Sets up timing, synchronization, etc. for real-
                                 time viewing

LINE 2/     CONFIGURATION DD     Sets up Configuration 1 - both displays driven
                                 by digital computer

LINE 3/     SAMPLE Ø             Sample A/D channel Ø and assign the value to the
LINE 4/     S1L                  variable S1L (Spring 1 Length Scale Factor)

LINE 5/     SAMPLE 1             Sample A/D channel 1 and assign the value to the
LINE 6/     S1W                  variable S1W (Spring 1 Width Scale Factor)

LINE 7/     SAMPLE 2             Sample A/D channel 2 and assign the value to the
LINE 10/    S1V                  variable S1V (Spring 1 Vertical Displacement)

LINE 11/    SAMPLE 3             Sample A/D channel 3 and assign the value to the
LINE 12/    M1V                  variable M1V (Mass 1 Vertical Displacement)

LINE 13/    TRANSFER             Transfer one point of the image from the source
LINE 14/    TOP                  table named TOP to the refreshed display and
LINE 15/    DISPLAY 2V           display in the vector mode

LINE 16/    CONTINUE             Continue the above initiated transfer until
                                 completed

LINE 17/    TRANSFER             Transfer one point of the image from the source
LINE 20/    MASS                 table named MASS to the refreshed display
LINE 21/    DISPLAY 2V           and display in the vector mode

LINE 22/    MOVE                 During the above initiated transfer, move the
LINE 23/    ZERO                 image in the X direction per the variable ZERO
LINE 24/    M1V                  (which has a value of Ø) and in the Y direction
                                 per M1V

LINE 25/    CONTINUE             Continue the above initiated transfer until
                                 completed

LINE 26/    TRANSFER             Transfer one point of the image from the source
LINE 27/    SPRING               table named SPRING to the refreshed display
LINE 30/    DISPLAY 2V           and display in the vector mode

LINE 31/    SCALE                During the above initiated transfer, scale the
LINE 32/    S1W                  image in the X direction per the variable S1W
LINE 33/    S1L                  and in the Y direction per S1L

LINE 34/    MOVE                 During the above initiated transfer, move the
LINE 35/    ZERO                 image in the X direction per the variable ZERO
LINE 36/    S1V                  and in the Y direction per S1V

LINE 37/    CONTINUE             Continue the above initiated transfer until
              .                  completed
              .
              .
            GOTO LINE 1
```

Figure 6—Illustrative example of AID programming

interest has been expressed by a group of students in tying the College's two wind tunnels into the PDP-9 for on-line control and data reduction. Once this has been accomplished it is but a small step to bring the thrill of direct on-line real-time experimentation into the classroom through use of the AID System. Another area of expansion under active consideration is a link to the IBM 360/50 at our computer center. This would open up new areas of application in which the problems are best cast in general or problem-oriented high-level languages such as FORTRAN, GPSS, NASAP, etc. In this mode of operation the classroom control unit would most likely be a typewriter-like keyboard. Still yet another area of expansion is in application programs resident in the PDP-9 itself. Work is under way in this area with program packages for deterministic and stochastic signal analysis (e.g., FFT algorithms, random number generators, etc.). In this as well as in the above cases, the structure of the AID Language will allow for efficient integration of the expansions into the AID System.

REFERENCES

1 R M BAECKER
Interactive computer-mediated animation
PhD Thesis Massachusetts Institute of Technology 1969

2 C CSURI
Real-time film animation
Proceedings of UAIDE Annual Meeting 1970

3 F GRACER M W BLASGEN
Karma: A system for storyboard animation
Proceedings of UAIDE Annual Meeting 1970

4 W H HUGGINS
Iconic communications
IEEE Transactions on Education November 1971

5 D C MARTIN
A different approach to classroom computer use
ACEUG Transactions Vol 1 No 1 January 1969

6 J W WILLHIDE
A system for real-time dynamic graphics
Proceedings of Fifth Hawaii International Conference on System Sciences 1972

GOLD—A Graphical On-Line Design System

by L. J. FRENCH and A. H. TEGER

RCA Laboratories
Princeton, New Jersey

INTRODUCTION

Computer programs do not "design." They blindly follow procedures that the computer programmer thought should solve a given class of problems. However, the methods of solution for many design problems are not *a priori* known. Actively inserting the human in the design process can add the elements of experience, insight, and creativity to the computer's ability to perform rote design procedures and repeated checking operations.

Graphic displays are ideally suited to this combination of computer power and human intellect if, and we must emphasize only if, the application deals with large quantities of data that must be absorbed or modified graphically. Teletypes or text displays are inadequate to deal with these pictorial problems. Full graphic displays are the only practical mechanism to provide rapid creation of a complex picture that can be quickly understood by the designer.

We have developed a general system for Graphical On-Line Design that operates stand-alone on a small computer, and yet supports a large, powerful data structure partitioned for local disc storage. The data structure, described in more detail later, is capable of storing relationships needed for computer-aided design, including non-graphic as well as graphic information.

The GOLD system has been applied to the design of integrated circuit masks. IC masks, especially for large arrays, contain huge amounts of graphical data that cannot be checked by the human designer without some display or plot. Furthermore, attempting to correct the artwork for these masks from a file of text plotting statements is an error-prone and time-consuming process. The interactive graphical editing possible with GOLD is alleviating these problems.

We designed GOLD to be a user-oriented interface between the computer and the designer. As such, the graphical language used to modify the artwork has to be easy to understand and to use by an integrated circuit designer; it must correspond to the natural thinking process that the designer would use to make corrections by hand. Yet the graphical language must also be general enough to apply to other application areas, and to deal with a hierarchical data structure smoothly enough so that the user can take advantage of its power without needing to understand how the structure is organized.

Perhaps the two most significant contributions of the work described herein are the creation of an efficient, hierarchical data structure, and of general graphical editing techniques that incorporate application dependent parts within the confines of a small stand-alone computer. General purpose graphical systems and graphical editing systems for IC masks exist, some of which are listed in the references, but the GOLD system represents an attempt to achieve the generality of large graphical systems and the practicality of smaller IC mask design systems on a small stand-alone computer.

The data structure described herein is ring type and hence similar to that of McCarthy,[8] Sutherland,[1] and Wiseman[2] yet is different in design in that it is specifically partitioned for secondary storage and is efficient in core utilization both in storage of the data structure and of the data structure functions themselves. The SELECT and LOCK editing functions are new and represent attempts at overcoming refresh display hardware limitations (flicker) and providing convenient techniques for rapidly protecting and interacting with various hierarchical levels in the data structure.

SYSTEM DESCRIPTION

GOLD hardware system

The large capital investment and operating costs for large computers and the advent of inexpensive, relatively powerful mini-computers convinced us to utilize

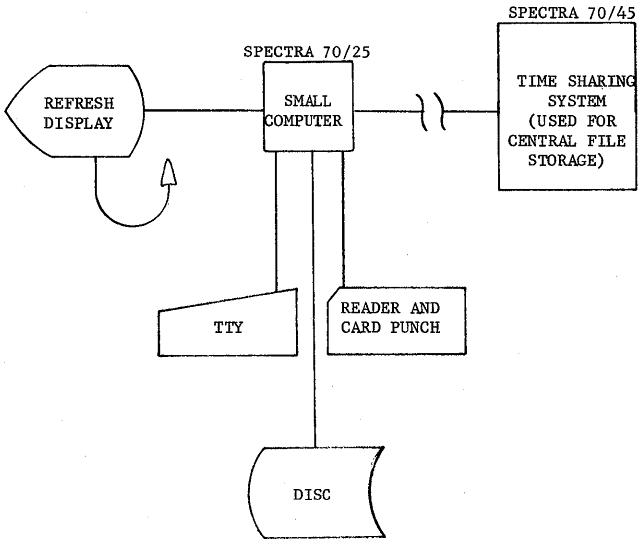


Figure 1—GOLD hardware system

a small computer in our interactive graphical design system. If such a system can perform a useful task, it might then be, and present evidence suggests that it is, economically justifiable.

The advantages of convenient and rapid interaction influenced our decision to utilize a refresh type display. A block diagram of the GOLD system is shown in Figure 1. The display is a refresh type, custom made by Information Displays, Inc. and is refreshed via a Spectra 70/25 computer having 32K bytes of memory with a 1.5 μsec cycle time. The input/output devices are a light pen, typewriter, card reader/punch, and a remote time sharing system. The disc is a single, 7¼ million byte capacity RCA 70/564 drive.

The philosophy of the GOLD system is that the fast secondary storage medium (disc unit in this case) can be utilized to provide access rates fast enough for rapid on-line interaction with a very large data base. The small computer can operate on a stand-alone basis, not simply as a display controller.

Computer-aided design system for integrated circuit masks

The GOLD system is compatible with other techniques for creating and editing integrated circuit artwork that have been developed at the RCA Solid State Division in Somerville, New Jersey. Two of these techniques are a digitizer system and a user-oriented mask description language, called PLOTS. The digitizer system captures a mask description by physically tracing a drawing point by point. The mask description obtained is stored in central files of a time sharing system. These files can be accessed by GOLD, where they can rapidly be displayed, edited, and stored either locally, or returned to the time sharing system.

PLOTS, also developed at RCA Solid State Division, is an artwork language used to describe drawings interactively on the time sharing system via a teletype. The files of PLOTS statements can similarly be accessed by GOLD for interactive graphical manipulation with the option of returning the modified files to the time shared system. An output program can convert either PLOTS or digitizer files to drive high accuracy plotters to obtain finished artwork. The compatibility that GOLD has with these two artwork systems gives a mask designer the flexibility of being able to select the best editing system for any part of a mask design.

GOLD software system

A block diagram of the GOLD software system is shown in Figure 2. Mask descriptions can be entered via the on-line typewriter, light pen, cards, or from a time sharing file. The mask description is processed by the input language compiler which checks syntax and produces object code for the Data Structure Compiler (DSC). The DSC builds a partitioned hierarchical data structure using a ring type language and a set of subroutines which provide techniques of creating and traversing partitioned ring type data structures. The

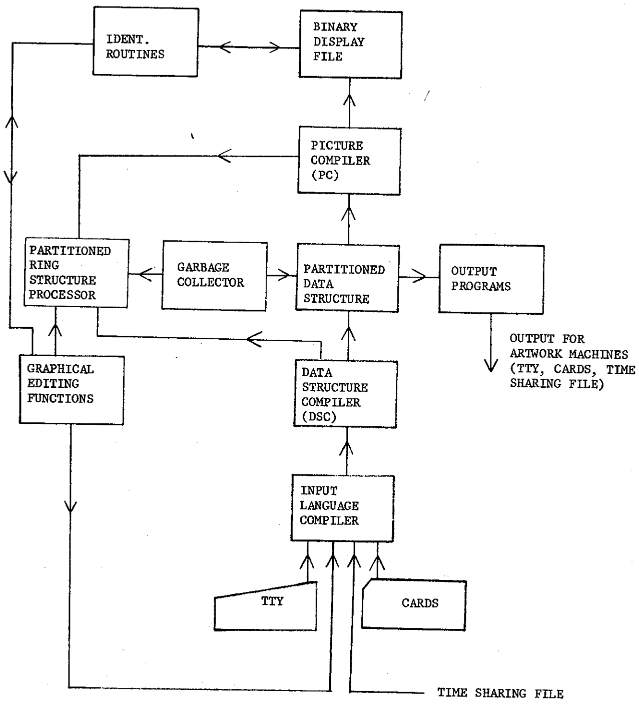

IDENT. ROUTINES
BINARY DISPLAY FILE
PICTURE COMPILER (PC)
PARTITIONED RING STRUCTURE PROCESSOR
GARBAGE COLLECTOR
PARTITIONED DATA STRUCTURE
OUTPUT PROGRAMS
OUTPUT FOR ARTWORK MACHINES (TTY, CARDS, TIME SHARING FILE)
GRAPHICAL EDITING FUNCTIONS
DATA STRUCTURE COMPILER (DSC)
INPUT LANGUAGE COMPILER
TTY
CARDS
TIME SHARING FILE

Figure 2—GOLD software system

partitioned data structure contains the accurate and hierarchical description of the application being undertaken and provides a general interface for all data manipulations.

The Picture Compiler (PC) is a recursive program that traverses the partitioned data structure and creates a binary display file within the confines of the user supplied specifications (see the display function in the third section). The binary display file is contained in part of main memory and is used to refresh the display to produce a picture. The picture compiler inserts into the binary display file markers that provide the hierarchical connection between the binary picture representation and its corresponding description in the data structure.

The light pen aids in graphical interaction by detecting light and interrupting the processor. The light pen interrupt is processed via the Identification Program which blinks the identified entity and provides the link to the actual representation of the entity in the data structure. The Graphical Editing Functions use the information supplied by the Identification Program to gain access to the data structure and all editing is performed on the data structure using the PRSP language and corresponding functions. Finally, the output programs traverse the data structure, again using PRSP, and provide an output mask description on either cards or in a time sharing file.

The data structure and data structure language

The data structure that has been designed and implemented within the GOLD system is a partitioned ring type data structure. It has the capability of containing 500K bytes of data which is partitioned into 128 segments in secondary storage. The data structure provides the capability of storing graphical information and hierarchical relationships. All mask coordinates are stored in a floating point notation created specifically for the Spectra 7Ø/25 and all floating point operations are performed via software due to the absence of hardware. Any graphical entity can be edited and entities can be combined into higher ordered entities within the data structure.

Any entity can be locked or selected, which in the data structure simply corresponds to a few bits being set in the proper entity. If an entity has been locked, editing will not be permitted on that entity. If an entity that is locked is pointed to by the light pen, the identification program automatically identifies the next higher unlocked entity. Thus, the lock function provides the graphics user the capability of controlling what can be edited and on what hierarchical level identification should initially occur. Each entity in the data structure has a bit which can be set to indicate that the entity has been selected. Presently this general data structure function has been employed only in displaying entities as discussed in the third section. Select is a general function which allows a user to select a class of data structure entities for any desired operation.

The data structure language (PRSP) provides a general language for creating and traversing partitioned ring type data structures. It is a new language based primarily on the work of Wiseman and Hiles of the University of Cambridge in England. PRSP is a macro language that expands into subroutine calls on 32 different data structure functions to provide full data manipulation capability with a common but well specified interface. Perhaps most noteworthy of all is that the data structure subroutines and the file management program occupy only $1ØØØ_{16}$ bytes of core. Core is at a premium in a small computer and the size of any program must be weighed very carefully against the power the program adds to the overall system. Each function in PRSP has been designed to provide the greatest capability in the least amount of core storage. In PRSP the programmer has the capability of partitioning his data structure on secondary storage to suit his particular circumstances. For the GOLD system a partitioning algorithm was selected to minimize disc I/O for operations on a specific mask level.

The file management routines in conjunction with PRSP provide rapid access to secondary storage for program overlays as well as for data structure operations. If a small computer is to handle 500K bytes of data structure and all the programs to provide a stand-alone system, the secondary storage retrieval time is crucial to the success or failure of such a system. In the GOLD system the average access time per segment (4120 bytes of information) is about 55 msec. This is fast considering that the disc rotation time for the raw information is 31 msec. PRSP can handle any number of data structure segments in core. However, present core size has limited this to two data structure segments. The file management routines calculate a "success factor" for each data structure segment in core. Whenever a data structure function requests a segment that is presently in core, the success factor is incremented by one. Whenever a segment is requested that is not in core, the least successful segment is overwritten by the new segment. In the fourth section the results of two file management algorithms are discussed.

Picture compilation

In the GOLD system the picture compiler compiles a picture as constrained by the user. Since the amount

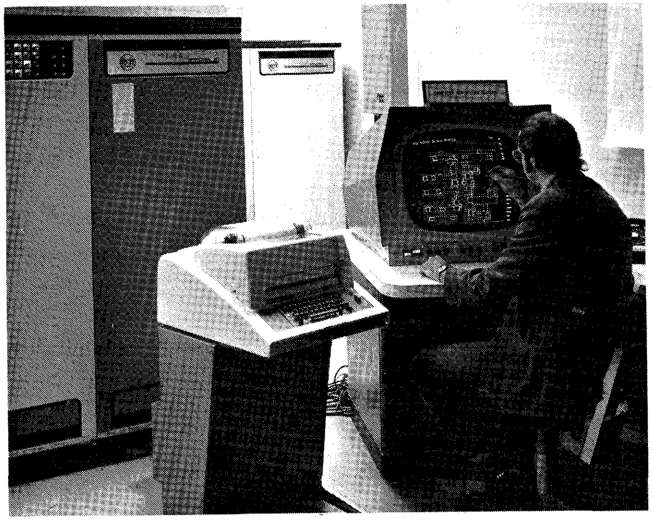


Figure 3—Display hardware layout

of data involved in a mask description may be roughly ½ million bytes, it is impossible to maintain the entire mask description in a display file in core. Thus, the user specifies what part of the mask he wishes to see and the picture compiler provides that specific display file by traversing the data structure. One of the functions the picture compiler performs is zoom (windowing). While our display does not have hardware zoom, software zoom would be required to provide a reasonably sized display file. The speed of the picture compiler is the most critical of all programs since it affects the interaction time of the system. An evaluation of the picture compiler program is contained in the fourth section.

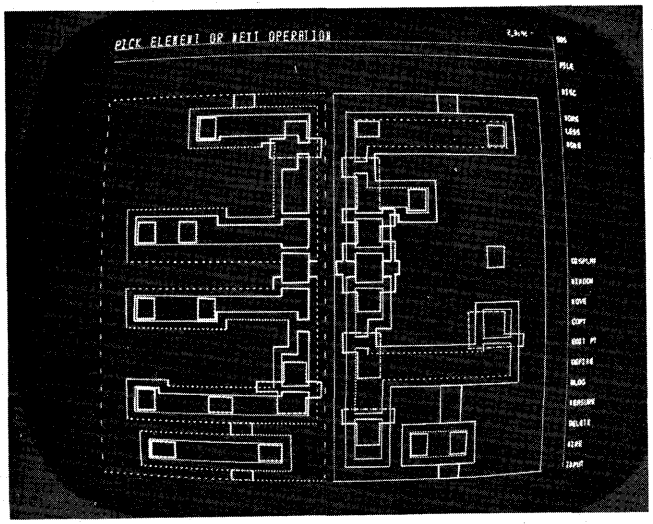


Figure 4—Display screen showning an IC design with 3 mask levels superimposed. There is no display flicker

GRAPHICAL CAPABILITIES

A user of the GOLD system wants to modify the artwork for a given set of masks by making additions or corrections to it. He therefore needs to access the files containing the artwork description, choose the portion to be displayed, and then be able to do the actual editing operations. GOLD provides a set of graphical tools to perform these functions. The light pen is used to pick operations, identify parts of the drawing, and perform many of the manipulations. The user inputs lengthy text messages such as personal file names or identification with the on-line typewriter. Figures 3, 4 and 5 show examples of display usage.

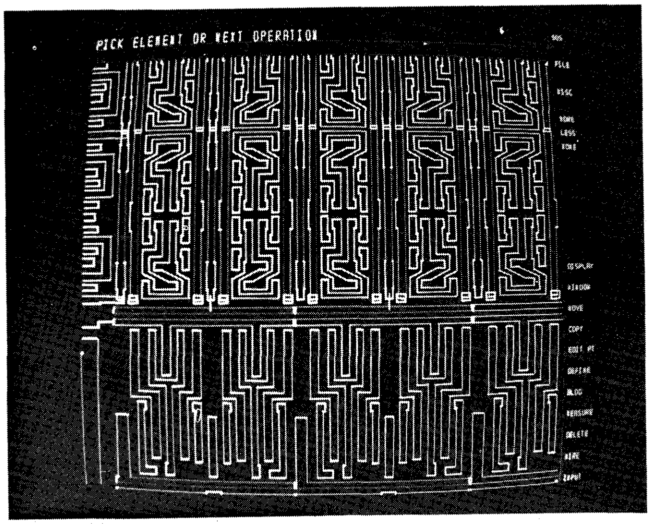


Figure 5—Display of part of a complex mask (zoom used). Flicker is evident

The display screen is divided into three functional areas. On the right side of the display, the menu of command choices is shown. These choices change as the user performs a given function so as to guarantee that only legal commands can be picked at any step in the process, and to provide detailed commands for each general function requested. At the top of the screen, system messages are displayed to guide the user in choosing the correct sequence of operations, and to issue error messages when needed. Error messages are blinked to attract the user's immediate attention; normal requests to indicate the next step are not blinked.

Beneath the message line is a line that can contain temporary information for the user, such as x,y data, and in which the user composes various light pen responses for the system. The third and largest functional area, occupying the central portion of the screen, displays the picture requested by the user. The actual editing changes are seen here.

File access

By pointing the light pen at the command FILE, the user has a choice of retrieving artwork files in various formats. For example, PLOTS files, consisting of "English-like" plotting statements, can be accessed via the local card reader or from remote files on the time sharing system. The system allows the user to define and repeatedly call graphical subroutines of any degree of complexity. These calls may be nested up to 10 levels deep in PLOTS, however, the nesting is unlimited in the data structure used by GOLD.

Another input format is the data structure itself. The user may store files on the local disc or on cards with all the structural information added during the graphical design session, and then retrieve the files for later enhancement.

Display creation

Having read in a file, a user has several tools for creating the desired display. He can specify which mask level should be shown, or request a combination of any number of mask levels. The line structure for each mask level can be individually chosen as either solid, dotted, dashed, or dot-dashed. Thus, he can check alignment of entities between mask levels, and still distinguish, by the line structures, which level each shape is on.

The WINDOW function permits a user to zoom in or out on any section of the artwork. To zoom in, he uses the light pen to position a tracking mark at the origin of the desired window. The absolute x,y value in mils of this coordinate is displayed as a numeric check. He draws a square with the light pen indicating the area he is interested in. When satisfied, he points to an OFF button attached to the tracking mark, and the area within the square is enlarged to full screen. Whichever mask levels and line structures were displayed before are retained after the windowing operation. The zoom feature permits a user to work on large drawings to whatever precision he desires by magnifying that section of the artwork and consequently overcoming the inherent inaccuracies of any graphical display. The zoom operation is performed on the data structure so that full precision is maintained in the display file. At any time, the user may point to a light button requesting full artwork size. This gives in effect a quick zoom out to display the full artwork.

The major problem in the GOLD system is display flicker, that is, the amount of information that can be displayed. Presently the system has memory space to display 2000 vectors while flicker occurs at about 800 vectors. The fact that new display hardware currently available is a factor of 2 or 3 faster than our display would aid in this problem but certainly would not solve it. The functions WINDOW and SELECT (described under Editing techniques) in some cases solve the problem by allowing the user to remove unnecessary parts of the drawing from the screen in order to eliminate confusing detail and reduce flicker. However, in other cases the user cannot eliminate parts (or enough parts) of the picture and needs to display 4000 to 5000 vectors without flicker. Some of the users who originally thought they could do little, if any, editing without seeing large quantities of the picture at a time, have since learned to think about mask design in a different fashion and have found they can do a great deal of work in a piecewise manner.

Editing techniques

All the graphical functions can operate on any entity displayed on the screen. The lowest level (atomic) entities are lines, simple polygons and orthogonal polygons. The system will treat a light pen hit on any vector on the screen as though the entire entity had been identified and will blink that entity. For example, a hit on a side of a complicated polygon will cause the entire polygon to blink, and the user, having visually verified that he does mean that polygon, can now modify the polygon via one of the graphic functions.

The general editing functions encompassed by the system are building block, move, copy, delete, rotate, measure, input and select.

1. BUILDING BLOCK

The power of graphical interaction lies in being able to create and operate on more complicated entities. The user, by pointing at the words BLDG BLOCK (a light button in the menu), can designate any number of atomic entities and previously defined building blocks as a new building block. The entities may be connected or unconnected, and may be from several different mask levels. The new building block is placed in the data structure to indicate its relationship to the rest of the artwork. Building blocks may be nested to any desired degree.

In order to use these building blocks, the light buttons MORE, LESS and NONE are part of the menu. On a light pen hit, the lowest level non-locked entity will blink on the screen. Pointing at MORE tells the system to move up to the next level of complexity in the data structure, i.e., the next higher building block. This can be repeated till the most complex building block, of which the original vector is a part, is blinking. Whatever is blinking at a given time can be operated on by any one of the graphical functions. LESS steps downward in complexity through the building blocks, and NONE stops everything from blinking.

2. MOVE

Any blinking entity (entity and building block will be used synonymously from this point on) can be moved using the light pen. A tracking mark appears on the reference point of the entity. By pointing at the tracking mark, the user moves the entire entity across the screen. The x,y location of the tracking mark in mils is shown at the top of the screen as a numeric check. The user can constrain the motion to x only, or y only, at any time. When the light pen is released, the entity "jumps" to the nearest user-defined grid location. This grid, which can be changed whenever desired, permits smooth tracking of entities, and yet precise positioning to any degree of coarseness. The user can also step the entity in increments of this grid horizontally or vertically.

The user may specify the absolute x,y position where the entity should be moved or he may give a numerical increment (up, down, right or left). In these cases the entity jumps to the new position and the absolute x,y location is again displayed on the screen.

When satisfied, the user points to 'OK' to finalize the change.

3. COPY

Any degree of complexity of building blocks may be copied. The user identifies the desired entity, checks that it is blinking, and requests a copy. The copy is drawn, slightly displaced from the original, and MOVE automatically called to operate on the copy.

4. ROTATE

The user may rotate certain entities in increments of 90°, and may mirror them about the x or y axis. All eight possibilities of mirroring and rotation are available.

5. DELETE

Any identified (blinking) entity may be deleted.

6. MEASURE

The tracking mark provides a numeric check on any x,y location on the screen. The optional imposed grid and x only, or y only, tracking constraints are the same as those described in MOVE.

7. INPUT

This function allows the user to create new atomic entities, namely lines, simple polygons or orthogonal polygons. The user specifies which mask level the new entity is to appear on, and a desired width if the entity is a line. Lines are drawn as a center line representation to reduce clutter on the screen, and can have an optional orthogonality constraint applied. Simple polygons may have sides at any angle, although the tracking mark can be used to make any given segment horizontal or vertical. The system will guarantee closure of both simple and orthogonal polygons, or give an error message if not possible. The special case of rectangles is handled by allowing the user to fix one corner of the new rectangle with the tracking mark, and then move the diagonally opposite corner (via the tracking mark) to draw in one motion all four sides. All x,y values are continually displayed at the top of the screen while these entities are being drawn.

8. SELECT

In order to reduce clutter and to minimize flicker on the display, the user can select any number of entities with the light pen. He then can choose to display *only* the selected entities, to blank *only* the selected entities, or to display all entities. The system reminds him of how many entities are selected at any given time. This feature permits work on complicated artwork at full size without destroying the sanity of the user.

File storage

During a design session, the entire data structure may be stored temporarily in any of four work areas on the disc. This allows the user to try different modifications of the job, and still be able to back up to any of these previous check points if desired.

After editing is complete, the file may be stored locally in data structure form, or converted back to PLOTS and stored on the time sharing system or cards. From the time sharing system, the files may be plotted on either of two high accuracy flat bed plotters to produce the final artwork for the masks.

SYSTEM EVALUATION

All editing functions in the GOLD system are performed on the data structure using PRSP and require no more than at most a few seconds. As explained in the second section, the interaction rate of the system is critically dependent upon the picture compiler. The picture compiler program extracts entities on the user specified mask levels, handles the SELECT and LOCK functions, performs scaling, rotation, all windowing, and the creation of a binary display file with embedded data structure markers. The picture compilation time varies in a rather complicated fashion from a few seconds to a maximum of about 1½ minutes with the average being about ½ minute. As an example, to create a display file of 800 vectors, the picture compiler requires 15 seconds.

A memory array designed at RCA Laboratories by J. Meyer and N. Kudrajashev using the GOLD system has been used as a benchmark to evaluate the operation of the system. The circuit is a 256-bit random access CMOS/SOS memory chip which measures 144×173 mils. The description consists of 7 mask levels and approximately 17,ØØØ points. To evaluate where the picture compile time is spent, measurements were made on the picture compilation of all mask levels with everything inside the window. In the actual operation of the GOLD system this picture compilation would terminate abruptly when the display file core space was exhausted. To overcome this, the picture compiler was temporarily patched such that the actual binary display file words were discarded. *Thus, this compilation represents a situation that cannot exist in practice but does give a worse case evaluation that is an order of magnitude worse than actual conditions.*

Using a file management algorithm which handles only one piece of data structure in core at a time, the picture compilation required a total of twenty minutes. We must stress that this is not a real case. The amount of information here (all parts of all mask levels) is far more than the user could visually understand or work with on a refresh display, and more than could fit in any display file.

The picture compile time distributes as follows:

DISC I/O

TOTAL DISC TIME = 2.71 MINUTES
= 13.6 PERCENT OF TOTAL TIME

TOTAL # OF DISC I/O = 2937

FLOATING POINT OPERATIONS
(software subroutines)

# ADD'S = 157,984	(1.46 msec/ADD)
# MULT'S = 60,225	(1.36 msec/MULT)
TOTAL # = 218,209	

TOTAL FLOATING POINT TIME
= 5.22 MINUTES
= 26.1 PERCENT OF TOTAL TIME

DATA STRUCTURE OPERATIONS

TOTAL # OPERATIONS (FUNCTION CALLS)
= 393,339

TOTAL D.S. OPERATION TIME
= 7.0 MINUTES
= 35 PERCENT OF TOTAL TIME

OTHER COMPUTE TIME
(EXCLUDING ABOVE)

= 5.07 MINUTES
= 25.3 PERCENT OF TOTAL TIME

The entry labeled "OTHER TIME" accounts for the decision making time of the picture compiler. The average instruction execution time of the 70/25 is approximately 20 μsec. Thus, in twenty minutes this is about 2.4×10^9 executed instructions. These numbers show two significant things. First, the amount of information contained in integrated circuit artwork is enormous. Second, it is important to notice that the disc I/O time, even with the very simplest of file management algorithms as used in this measurement, is relatively small. This is impressive considering that the entire GOLD system operates on both the data structure and all programs in a partitioned (virtual memory) environment.

The same picture compilation was rerun using a file management algorithm that manipulates two data structure segments in core at a time. This algorithm reduced the number of *disc I/O's to 759 and a total disc time of .7 minutes.* This reduced the overall compile time to 18 minutes, such that the *disc I/O time was only 3.9 percent of the total compilation time.* While this algorithm reduces the disc I/O time significantly, the overall effect on the system time is small. The conclusion to be drawn from this is that the GOLD system is not disc limited as one might anticipate from a virtual memory system. Part of this is due to the design of the data structure, part to the design of the file management routines, part to the disc file organization, and part due to the fact that the 7Ø/25 is a moderate, if not slow, computer. To explain the latter point in more detail, it is possible that many of the modern mini-computers could decrease the average instruction execution time by roughly an order of magnitude. Some have hardware push down stacks which would significantly decrease the time, perhaps even more than the ten to one reduction in the faster instruction set. In any case a faster instruction speed would decrease the PRSP and OTHER TIME by a factor of ten but would have no effect on the disc I/O time. The times for the various items would then be as follows:

DISC I/O

# READS=759	UNCHANGED
TOTAL DISC TIME =.71 MINUTES=12.8 PERCENT	

FLOATING POINT OPERATIONS

# ADD'S=157,984	(1 msec/ADD)
# MULT'S= 60,225	(1 msec/MULT)

TOTAL #=218,209

TOTAL F.P. TIME
=3.62 MINUTES=65.3 PERCENT

PRSP FUNCTIONS

TOTAL # OPERATIONS=393,339

TOTAL TIME
=.7 MINUTES=12.8 PERCENT

OTHER TIME
=.51 MINUTES=9.1 PERCENT

TOTAL COMPILATION TIME
=5.54 MINUTES

These last figures show that if the computer was a modern mini-computer, the disc I/O time would become a somewhat more significant part of the system time but would still be relatively unimportant. Another way of viewing this is that if the system described above had infinite core, such that the disc I/O time was zero, it would only be 12.8 percent (disc I/O part) faster.

These figures also show the usefulness of floating point hardware. If the system described above had 10 μsec floating point hardware instructions, the total floating point time would be 2.18 seconds instead of the 3.62 minutes required by the software floating point operations. This would increase the overall speed of the system by approximately 65 percent and reduce the total compilation time to less than 2 minutes. Thus, a modern mini-computer with floating point hardware would have more impact than a mini-computer with a large memory capacity.

One of the problems generally encountered in the utilization of list type data structures is the large amount of overhead storage required due to pointers, etc. In the GOLD system significant effort has gone into creating an efficient data structure while maintaining the advantages of list type structures. In many list type data structures it is not uncommon for the data structure to require 10 times more storage than the raw data alone (canonical form). The GOLD data structure, for the benchmark integrated circuit previously described, requires 71,677 bytes of storage. The canonical raw data form (non-structured) requires 25,158 bytes of storage. This is a ratio of 2.8 to 1, which shows that the data structure is rather efficient in storage utilization. Recent work in this area has resulted in relatively simple techniques (simple in implementation) which will reduce the storage from 71,677 bytes to approximately 56,000 bytes, and would make the ratio 2.2 to 1. The size of data structures at this point presents little problem to the GOLD system. The mask description for the benchmark integrated circuit is one of the more complicated circuits being designed yet only uses about 15 percent of the available data structure storage.

SUMMARY

The GOLD system has been utilized on an experimental basis for about one year. Most users have found the system easy to learn, and to use. However, all users have required some training, to varying degrees, to utilize the system's more sophisticated capabilities.

We have overcome the basic inaccuracies of the display hardware by providing continual numeric checks, a quantized user grid, and zooming ability. The power of GOLD is illustrated by building block, which gives the ability to work with functional units as a whole, and to define new functional units dynamically. Thus, the user can, for example, apply all the graphical manipulations to a line or to the transistor it belongs to, or to the circuit the transistor belongs to, etc.

The system offers very rapid and convenient interaction and is compatible with all other parts of the overall CAD system for integrated circuit artwork. We believe the GOLD system proves that it is possible and economically desirable to utilize a small computer on a stand-alone basis to provide a large, sophisticated, hierarchical data structure and graphical system.

ACKNOWLEDGMENTS

The authors wish to express their appreciation for the programming contributions to GOLD made by M. Gilchrist, J. Grabowski, G. Held, F. Larkin, C. Neroth, G. Payor, and H. Schnitzler. N. Wiseman and J. Hiles of Cambridge University, England, aided in the original version of the data structure routines, and M. Lewin designed and debugged the computer-display hardware interface. The cooperation of J. Miller and D. Ressler contributed to making GOLD compatible with RCA Solid State Division's artwork systems. Finally the support of G. B. Herzog and B. Lechner of RCA Laboratories made this work possible.

REFERENCES

1 I E SUTHERLAND
Sketchpad—A man-machine graphical communication system
Lincoln Lab Technical Report #296 30 Jan 1963

2 N E WISEMAN J O HILES
A ring structure processor for a small computer
the Computer Journal Vol 10 No 4 Feb 1968

3 T E JOHNSON
A mass storage relational data structure
MIT Dept of Architecture Preprint No 866

4 D E KNUTH
Fundamental algorithms
Addison and Wesley Vol 1 1968

5 D T ROSS C G FELDMANN
Project MAC
MIT Report MAC-TR-4 May 6 1964

6 M H LEWIN
An introduction to computer graphic terminals
Proc IEEE Sept 1967

7 ADAMS ASSOCIATES
The computer display review
Vol 1 July 1966

8 J McCARTHY et al
LISP 1.5 programmer's manual
MIT August 17 1962

9 N E WISEMAN
A note on compiling display file from a data structure
University Mathematical Laboratory Cambridge Mass
March 1968

10 L G ROBERTS
Graphical communication and control languages
Lincoln Lab Report

11 W R SUTHERLAND
The coral language and data structure
Lincoln Lab Techn Report 405 Jan 1966

12 L G ROBERTS
Homogeneous matrix representation and manipulation of N-dimensional constructs
Lincoln Lab Report MS-1405 May 1965

13 L G ROBERTS
A graphical service system with variable syntax
Lincoln Lab Report Vol 9 No 3 March 1966

14 *Development of on-line system for computer-aided design of integral circuits*
Norden Division of United Aircraft Corp
Report AFML-TR-67-342 Vol 1 Oct 1967

15 W R SUTHERLAND
On-line graphical specification of computer procedures
Lincoln Lab Techn Report 405 23 May 1966

16 W H NINKE
Graphic 1—A remote graphical display console system
Bell Telephone System Tech Publication 1965

17 *Graphics*
Lincoln Laboratory Semiannual Technical Summary
31 May 1969

18 J C GRAY
Compound data structure for computer-aided design
ACM Proceedings 1967

19 T E JOHNSON
Sketchpad III, a computer program for drawing in three dimensions
SJCC 1963

20 I E SUTHERLAND
Computer graphics
Datamation May 1966

21 N LINDGREN
Human factors in engineering
IEEE Spectrum April 1966

22 L J FRENCH
An interactive graphical debugging system
Design Automation Workshop June 1970

23 J C MILLER C M WINE
A simple display for characters and graphics
IEEE Transactions on Computers Vol C-17 No 5 May 1968

24 L J FRENCH
A partitioned data structure system
Ph D Thesis Columbia University June 1971

25 F K RICHARDSON et al
An interactive, graphical system for the design of photomasks
NEREM Nov 1970

26 L H HAZLET
I am an integrated circuit design engineer
Electronic Design News June 1969

27 A SPITALNY M J GOLDBERG
On-line graphics applied to layout design of integrated circuits
IEEE Proc Nov 1967

28 D K LYNN
Computer-aided layout system for integrated circuits
IEEE Transactions on Circuit Theory Jan 1971

29 J S ZUCKER
Graphical layout system for integrated circuit mask design
IEEE Transactions on Circuit Theory Jan 1971

30 J F JARVIS
Interactive graphics mask design program
ISSCC Feb 1971

Establishing lower bounds on algorithms—A survey

by E. M. REINGOLD

University of Illinois
Urbana, Illinois

INTRODUCTION

Algorithms for various computations have been known and studied for centuries, but it is only recently that much theoretical attention has been devoted to the analysis of algorithms. Turing machines and recursive functions were the first approaches, but these models, which provide much interesting mathematics, do not look at the problem from a practical standpoint. In "real" computing, no one uses Turing machines to evaluate polynomials or to multiply matrices, and little of practical significance is obtained from that approach. On the other hand, recent work has led to more realistic models and, correspondingly, to more practical results. Most of the results cannot be considered to be truly practical, but, all of them were motivated by practical considerations.

This survey is concerned with efforts to establish lower bounds on the number of operations required to solve various practically inspired problems; in particular we discuss the problems of sorting, searching, merging, root finding, polynomial evaluation, matrix multiplication, and many others. No theorems will be rigorously proved; for some the idea of the proof will be presented, and most will only be stated. The reader is urged to pursue in the literature the details of any topics which interest him.

In the establishment of lower bounds on algorithms we must consider the following questions:

- What function or class of functions is to be computed?
- What class of algorithms is allowed?
- With what are we measuring lower bounds?

The answers to the last two of these questions are inherently interwoven with the answer to the first question. In analyzing sorting we will consider different things important than in analyzing matrix multiplication, and so in each case we will allow different kinds of algorithms and we will measure their efficiency in different ways.

Even for a specific answer to the first question, how the efficiency of an algorithm should be measured is not obvious. Ideally, we would like to assign a realistic cost to every operation performed; such a model usually makes the establishment of lower bounds too difficult. To simplify the problem, we isolate the "key" operations and ignore all others. There are two ways to count the operations used by an algorithm: the number used on the worst case input or the expected number used on a random input, assuming some distribution of the inputs. An algorithm is *minimax optimal* or *worst case optimal* if no algorithm is more efficient in the worst case; an algorithm is *minimean optimal* or *average case optimal* if no algorithm is more efficient in the average case.

It should be noted that some of the results discussed here have never been formally published, but have become a part of the "folklore" of the area; in such cases the citation will be to the place they first found their way into print—usually a textbook. Moreover, this survey is not complete: to include every known result would give the paper undue length; in addition, many results undoubtedly remain unnoticed, buried in journals, technical reports, and unpublished manuscripts.

NOTATION

The floor and ceiling operations are defined as usual: $\lfloor x \rfloor$ is the greatest integer less than or equal to x, and $\lceil x \rceil$ is the least integer greater than or equal to x. We use the standard notation for the order of magnitude of a function: $f(n) = 0(g(n))$ if there is a constant $k > 0$ such that

$$\limsup_{n\to\infty} \frac{f(n)}{g(n)} = k.$$

If the limit

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = k$$

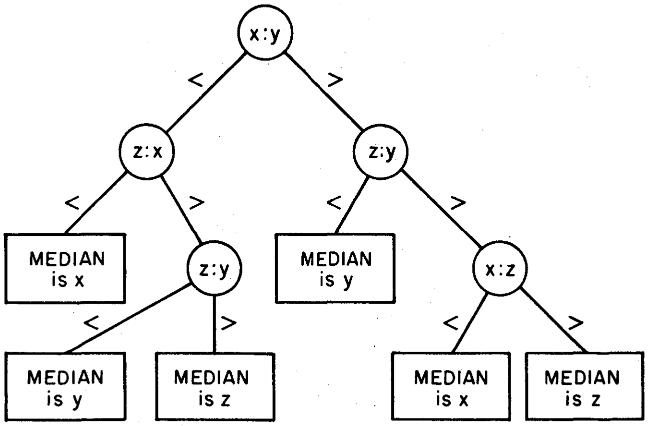


Figure 1—An extended binary tree representing the computation of the median of three numbers x, y, and z

exists and $k=1$, we say that $f(n)$ is asymptotic to $g(n)$, written $f(n) \sim g(n)$; if $k=0$ then we say that $f(n)=0(g(n))$.

The symbol "lg" is used to represent logarithms base 2, while "ln" is used for natural logarithms. We use "log" as the generic term when the base is unimportant.

COMBINATORIAL PROBLEMS

In studying the complexity of algorithms for combinatorial problems such as sorting, finding the median, merging, and others, we will allow algorithms to have only pairwise, errorless comparisons and no other operations. It is convenient to represent the algorithms as *extended binary trees* (like flowcharts), an example of which is shown in Figure 1. Each *internal node*, drawn as a circle, represents a comparison between two numbers (inputs or constants) and each *external node*, drawn as a rectangle, represents a possible outcome of the algorithm. Thus Figure 1 is the tree corresponding to an algorithm to find the median of three numbers x, y, and z.

The *height* of the tree corresponds to the number of comparisons used in the worst case. The *external path length* of a tree is defined as the sum of the distances from the root to each external node; thus if there are n external nodes, the external path length is n times the average number of compares, assuming all outcomes are equally probable. Both the worst case and the average case provide reasonable measures of efficiency. The following are useful:

Lemma 1:

The minimum external path length of an extended binary tree with n external nodes is $n\lceil \lg n \rceil + n - 2^{\lceil \lg n \rceil}$.

Lemma 2:

The minimum height of an extended binary tree with n external nodes is $\lceil \lg n \rceil$.

A complete discussion of most of the results discussed below can be found in Knuth.[Kn72]

Sorting

Suppose we are given a set $\{x_1, \ldots, x_n\}$; what is the minimum number of comparisons required to determine the permutation π so that $x_{\pi(1)} < x_{\pi(2)} < \cdots < x_{\pi(n)}$? Let $S(n)$ denote the minimum number of comparisons needed to rank n distinct inputs according to some ordering, then we have

Theorem 1:

$\lceil \lg n! \rceil \leq S(n) \leq 1 + n\lfloor \lg n \rfloor$ and thus using Stirling's approximation we have that $S(n) \sim n\lfloor \lg n \rfloor$.

The upper bound follows from some of the better algorithms for sorting, for example, the *binary insertion sort* first observed by Steinhaus[Stei50] before the advent of computer sorting. The lower bound is derived from Lemma 2 and the observation that each permutation of the n inputs must cause termination at a different external node of the tree. This is a "standard" information theoretic argument which says that given k outputs at least $\lceil \lg k \rceil$ binary decisions are required to distinguish between them. This argument appears to have been discovered independently by several authors but first appeared in Steinhaus.[Stei58]

There has been considerable work in refining the upper bound. Most notably, Ford and Johnson[Fo59] have developed a method of sorting which Hadian[Ha69a] showed required

$$\sum_{k=1}^{n} \left\lceil \lg \frac{3}{4} k \right\rceil$$

comparisons to sort n inputs in the worst case. Knuth[Kn72] has called this the *merge-insertion sort.* Comparing Hadian's result with Theorem 1 we find that the merge-insertion sort is optimal for $n \leq 11$ and $n=20, 21$ but that it requires more than $\lceil \lg n! \rceil$ comparisons for other values of n. Wells,[We65] using a computer, has shown that it is also optimal when $n=12$. Summarizing, we have

Theorem 2:

$$S(n) = \sum_{k=1}^{n} \left\lceil \lg \frac{3}{4} k \right\rceil \quad \text{for} \quad n \leq 12 \quad \text{and} \quad n=20, 21.$$

Define $\bar{S}(n)$ to be the minimum average number of comparisons required to sort n items. As before, we must have $n!$ external nodes and so by Lemma 1

$$\bar{S}(n) \geq \frac{1}{n}\,(n\lceil \lg n\rceil + n - 2^{\lceil \lg n\rceil}).$$

Letting $\lceil \lg n\rceil = \lg n + \theta$, $0 \leq \theta < 1$ this becomes

$$\bar{S}(n) \geq \lg n + 1 + \theta - 2^{\theta}, \qquad 0 \leq \theta < 1.$$

Since in $[0, 1]$ the function $1+\theta-2^{\theta}$ has a range of $[0, .08^{+}]$ we have

Theorem 3:

$$\bar{S}(n) \geq \lg n! + 0(1) = n \lg n - n/\ln 2 + 0(\log n).$$

Theorem 3 was first observed by Gleason[Gl56] and first published by Kislicyn.[Kis62,Kis63]

Searching

If we are given a sorted set $x_1 < x_2 < \cdots < x_n$, how difficult is it to determine in which of the $n+1$ ranges y lies? Let $s(n)$ be the minimum number of comparisons to do the searching. The binary search algorithm, first noted by Steinhaus[Stei50] requires $\lfloor \lg n\rfloor + 1$ comparisons. Applying Lemma 2, we see that $\lceil \lg(n+1)\rceil$ comparisons is a lower bound; since $\lceil \lg(n+1)\rceil = \lfloor \lg n\rfloor + 1$, we know that binary search is optimal in the worst case.

Sandelius[Sa61] noted that binary search is also optimal in the average case: applying Lemma 1 as before we see that the minimum possible average is $\lg(n+1)+0(1)$ and binary search achieves that bound.

A somewhat related problem is the discovery of the single counterfeit coin, either heavier or lighter, in a group of n coins; this problem is well known in the literature of recreational mathematics. One is usually allowed to use a balance scale and hence the comparisons are of linear functions, over $\{-1, 0, 1\}$, of the inputs rather than just pairwise comparisons. One optimality result for this problem is due to Smith[Sm47] and he has shown that when $n = (3^k - 1)/2$, k such comparisons are necessary and sufficient.

Merging

Given two sorted sets $A_1 < A_2 < \cdots < A_m$ and $B_1 < B_2 < \cdots < B_n$ (all distinct) what is the best way to merge these into a single sorted set $x_1 < x_2 < \cdots < x_{m+n}$? Since the $n+m$ elements are all distinct, there are $\binom{m+n}{m}$ ways the A's may appear among the B's. Thus by Lemma 2, if $M(n, m)$ is the minimum number of comparisons required to do the merging,

$$M(n, m) \geq \lceil \lg \tbinom{m+n}{m} \rceil.$$

A simple upper bound on $M(n, m)$ is $m+n-1$ since the "usual" merging algorithm outputs at least one element for each comparison; the last element requires no comparisons. Hwang and Lin[Kn72] have developed a much better merging algorithm which requires

$$m + \left\lfloor \frac{n}{2^t} \right\rfloor - 1 + tm \quad \text{where} \quad m \leq n \quad \text{and} \quad t = \left\lfloor \lg \frac{n}{m} \right\rfloor$$

since this is less than $\lceil \lg \binom{m+n}{m} \rceil + \min(m, n)$ we have

Theorem 4:

$$\lceil \lg \tbinom{m+n}{m} \rceil + \min(m, n) \geq M(n, m) \geq \lceil \lg \tbinom{m+n}{m} \rceil.$$

When $m = 1$, merging becomes a simple search and so

$$M(1, n) = \lceil \lg(n+1) \rceil.$$

When $m = 2$, the analysis of merging is quite difficult; Graham[Gr71] and Hwang and Lin[Hw71] have independently shown that

$$M(2, n) = \left\lceil \lg \frac{7}{12}\,(n+1) \right\rceil + \left\lceil \lg \frac{14}{17}\,(n+1) \right\rceil.$$

By the construction of what Knuth[Kn72] calls an *oracle*, Karp and Graham independently showed that

$$M(n, n) = 2n - 1.$$

An oracle is a hypothetical device which constructs a worst case for any possible algorithm; in other words, it acts as an adversary to the algorithm, forcing the algorithm to do the maximum possible work. The oracle in this case is $A_i < B_j$ if and only if $i < j$.

Selection

Suppose that there are n inputs and we want to know which is the k^{th} best in a ranking; Hadian and Sobel[Ha69b] have termed this the "selection problem." On the other hand, if we wish to know the k best and their ranking, they call this the "ordering problem."[Ha70] When $k=1$ and $k=2$, the ordering and selection problems coincide and a minimal solution for one is also a minimal solution for the other.

The case $k=1$ is trivial and a simple induction argument shows that $n-1$ comparisons are required. Rabin[Ra71] has shown that in the case $k=1$, that is, computing the maximum, $n-1$ comparisons are necessary even if comparisons may be made between any analytic functions of the inputs. For $k=2$, the problem of determining the minimal number of comparisons was first posed by Steinhaus in 1929. Schreier[Schr32] first stated the solution to this problem, but with an incorrect proof. Slupecki[Sl51] gave another incorrect proof. The first correct proof was given by Kislicyn[Kis64] by means of an oracle; he proved that

$n-2+\lceil \lg n \rceil$ comparisons were necessary and sufficient to find the second largest element in a set.

Recently, Blum, Floyd, Pratt, Rivest and Tarjan (personal communication) have developed a remarkable algorithm which finds the k^{th} largest element in a set of n elements in $0(n)$ comparisons in the worst case, regardless of k. Prior to the development of this algorithm it was commonly conjectured that the median of a set of n elements could not be computed in fewer than $0(n \log n)$ comparisons in the worst case. It was known[Van70] that it could be done by an algorithm in which the *expected* number of comparisons was $0(n)$; but in the worst case this algorithm requires more than $0(n)$ comparisons.

Summarizing these results, let $V_k(n)$ be the smallest number of comparisons required to find the kth of n elements, then we have

Theorem 5:

(a) $V_1(n)=n-1$
(b) $V_2(n)=n-2+\lceil \lg n \rceil$
(c) $V_k(n)=0(n)$ for all k.

Pohl[Po69] approached the selection problem in an entirely different way, showing that at least $\min(k, n-k+1)$ storage locations are required to determine the kth of n elements.

ALGEBRAIC PROBLEMS

In this section we will discuss the minimum number of arithmetic operations required to compute various functions. The arithmetic operations we will allow are addition, subtraction, multiplication, and division; no other operations (comparisons, exponentiation, etc.) will be allowed. To make an analysis, we need a precise definition of what algorithms are allowed.

Let $\circ$ denote any of the arithmetic operations addition, subtraction, multiplication or division. A *scheme* is defined as a sequence of operations

$$P_i = Q_i \circ R_i \qquad i=1,2,\ldots,m$$

where each Q_i and R_i is either a constant, an input value, or a P_j where $j<i$.

Polynomial evaluation

Suppose we are given a number x and asked to compute x^n for a fixed n, by a scheme as described above; what is the minimum number of steps required? Starting with a sufficiently large x, we can prove by induction that after k steps of a scheme the largest number obtainable is x^{2^k} which is computed by squaring x, squaring the result, and so on. Thus we must have

$$x^{2^k} \geq x^n$$

and hence

$$k \geq \lceil \lg n \rceil.$$

Thus at least $\lceil \lg n \rceil$ multiplications are required to compute x^n from x. Proving that this minimum is asymptotically achievable is more difficult. The following theorem is due to Brauer;[Br39,Kn69] Val'skii[Va159] arrived independently at the same result.

Theorem 6:

Let $m(n)$ be the least number of multiplications required to compute x^n for given values of x, then $m(n) \sim \lceil \lg n \rceil$.

This problem readily generalizes to: What is the smallest number of arithmetic operations needed to evaluate the n^{th} degree polynomial

$$f(x)=a_0+a_1x+\cdots+a_nx^n \qquad a_n \neq 0,$$

for given values of x? Ostrowski[Os54] was the first to suggest that this problem be analyzed, and he gave results for quadratic and cubic polynomials.

When it is known *a priori* that the values of x given will be equally spaced, a method using finite differences might be most convenient; however, we will assume that the values of x will be arbitrary. The method usually used is Horner's method:

$$f_0 = a_n$$
$$f_{r+1} = xf_r + a_{n-r-1} \qquad r=0,1,\ldots,n-1,$$

which requires n additions and n multiplications. Can this be improved? One can easily find specific polynomials for which there is a better method; for example

$$f(x)=1+x+2x^2+x^3+x^4$$

which can be evaluated in two multiplications and two additions by proceeding as follows:

$$x^2,\ x^2+1,\ (x^2+1)((x^2+1)+x).$$

However, we want a more general method, one which will work for all polynomials and for all values of x. Clearly Horner's method is a valid scheme for polynomial evaluation, and in fact, Horner's method is optimal, in the sense that it requires the fewest operations necessary in the type of scheme allowed, for we have:

Theorem 7:

Any scheme which can evaluate an arbitrary n^{th} degree polynomial has at least n additions/subtractions and n multiplications/divisions.

In this theorem, the necessity of the n additions/subtractions was shown by Belaga[Be58,Be61,Kn69] while the necessity of the n multiplications/divisions is due to Pan.[Pan66] Not only is Horner's method optimal, but it is uniquely optimal: Borodin[Bo71] has shown that any scheme using only those $2n$ operations is essentially Horner's method.

A different approach can be taken if a large number of values $P(x)$ are required. Consider the following example, due to Todd.[Tod55] We want to evaluate the polynomial

$$P=x^6+Ax^5+Bx^4+Cx^3+Dx^2+Ex+F.$$

Define the following polynomials

$$P_1=x^2+ax=x(a+x)$$

$$P_2=(P_1+x+b)(P_1+c)$$

$$P_3=(P_2+d)(P_1+e)$$

and determine a, b, c, d, e, and f such that $P=P_3+f$. This can be done by the solution of linear equations and a single quadratic equation. Once these equations have been solved, P can be evaluated using only three multiplications and seven additions using the sequence

$$P_1,\ P_2,\ P_3,\ P=P_3+f,$$

a savings of three multiplications at the expense of one addition and some "preconditioning" of the coefficients. Since multiplication is usually much slower than addition and since we sometimes want the same polynomial evaluated at many arbitrarily spaced points, the above method can represent a significant improvement over Horner's method.

A scheme with preconditioning is formally defined as a scheme in which the Q_i and R_i are, in addition, allowed to be any real functions of the coefficients of the polynomial to be evaluated. The scheme due to Todd above, is an example of such preconditioning. The idea of preconditioning is due to Motzkin[Mot55a] and he showed that if a scheme with preconditioning computes all n^{th} degree polynomials then it contains at least $\lfloor (n+1)/2 \rfloor$ multiplications.[Kn69] Combining this result with the full strength of the result of Belaga used in Theorem 7, we have

Theorem 8:

Any scheme with preconditioning which can evaluate an arbitrary n^{th} degree polynomial has at least $\lfloor (n+1)/2 \rfloor$ multiplications and at least n additions/subtractions.

Much effort has been spent to find a scheme with preconditioning which attains the lower bounds of Theorem 8. Early papers by Motzkin[Mot55a] and Knuth[Kn62] gave methods which evaluate polynomials of degrees four, five, and six in $\lfloor (n+1)/2 \rfloor+1$ multiplications and $n+1$ additions; Pan[Pan61,Pan65] has given similar methods for $n \leq 12$. In each of these cases, the methods are applicable only for a particular value of n. Pan[Pan61] gives a method valid for $n \geq 2$ which requires $\lfloor (n+1)/2 \rfloor+1$ multiplications and $n+2$ additions/subtractions and in Reference Pan59 he gives a method for $n \geq 5$ for which $\lfloor n/2 \rfloor+2$ multiplications and $n+1$ additions/subtractions are needed. For $n \geq 3$, Knuth[Kn62] gives a method using $n+1$ additions/subtractions and the number of multiplications varies between $\lfloor (n+1)/2 \rfloor+1$ and approximately $\frac{3}{4}n$. Belaga[Be61] proved that $\lfloor (n+1)/2 \rfloor+1$ multiplications and $n+1$ additions suffice to evaluate any nth degree polynomial, but these operations may involve complex numbers. Finally, Eve[Ev64] modified Knuth's method to give a method requiring $\lfloor n/2 \rfloor+2$ multiplications and n additions/subtractions, all of which involve only real numbers. The preconditioning in Eve's algorithm is, unfortunately, irrational; Rabin and Winograd (personal communication) have developed a method in which the preconditioning involves only rational operations, but his method then requires about $\lceil n/2+\lg n \rceil$ multiplications to evaluate a polynomial.

The best general algorithm for polynomial evaluation, Eve's, requires only the minimum number of additions/subtractions, however, it unfortunately requires one more than the minimum number of multiplications. It is known[Kn69] that when n is odd both of these lower bounds cannot be simultaneously achieved, and a similar result holds when $n = 4$ and $n = 6$. There is no known general algorithm using $\lfloor n/2 \rfloor+1$ multiplications when n—8, although such methods are known for $n=4$, 6, and 8, where the algorithms require one or two extra addition/subtraction operations.[Kn69,Pan62]

The above optimality theorems show that no one method will work for all polynomials of degree n unless it has a certain minimum number of operations, but there are some "special" polynomials which can be evaluated far more rapidly, for example, a x^{16} requires only five multiplications and no additions, instead of the minimums given by Theorems 7 and 8. There are "few" such polynomials for Belaga[Be61] has shown

Theorem 9:

The set of n^{th} degree polynomials which can be evaluated by schemes with preconditioning in fewer operations than specified in Theorem 7 has Lebesgue measure zero in the space of all nth degree polynomials.

Pan[Pan62] has proved a similar result for schemes without preconditioning.

Most of these results have been generalized to polynomials of many variables and to rational functions. In particular, such results are given in References Be58, Mot55b, Os54, and Pan62. Some of the results have also been obtained for the problem of simultaneously evaluating several polynomials in the same variable[Pan66] and for the specific case of the simultaneous evaluation of a polynomial and its first derivative.[Kn69, Mu71a]

Other activity in this area has concerned the analysis of polynomial evaluation when an arbitrary number of arithmetic operations can be performed in parallel; see, for example, References Ma71 and Mu71b. Also of interest is polynomial evaluation when the coefficients are rational; Paterson and Stockmeyer[Pat71] have shown that $0(\sqrt{n})$ operations are necessary and sufficient for the evaluation of an nth degree polynomial.

Linear algebra

By generalizing the notions in the above results to arbitrary fields, Winograd[Wi70] proved some very elegant theorems. Let F be a field and let $x_1, x_2, \ldots, x_n$ be a set of variables. The question then becomes, what is the minimum number of field operations needed to compute the m field elements

$$\Psi_i \in F(x_1, \ldots, x_n) \qquad i=1, \ldots, m.$$

Winograd gave a very general definition of a scheme without preconditioning, and considered only the number of multiplications/divisions. He showed:

Theorem 10:

Let Φ be an m by n matrix whose elements are in the field F, let ϕ be an m vector of elements in F, and let x denote the n column vector $(x_1, \ldots, x_n)$ so that

$$\Phi x + \phi \in F(x_1, \ldots, x_n)^m.$$

If there are u column vectors in Φ such that no nontrivial linear combination of them (over F) is in F^m, then any scheme, without preconditioning, computing $\Phi x+\phi$ requires at least u multiplications/divisions.

Pan's result on the number of multiplications needed for polynomial evaluation without preconditioning (part of Theorem 7) follows from this theorem as a corollary, for here Φ is the 1 by $n+1$ matrix

$$(1, x, x^2, \ldots, x^n)$$

and the columns of Φ are all linearly independent so that $u=n$.

We also have another corollary:

Theorem 11:

Let X be a p by q matrix and let y be a q column vector. Then to compute Xy requires at least pq multiplications/divisions and so the ordinary method of computing Xy minimizes the number of multiplications/divisions.

This follows from Theorem 10 by defining

$$\Phi_{ij} = \begin{cases} y_k \text{ if } j=iq+k & 1 \leq k \leq q \\ 0 \text{ otherwise} \end{cases}$$

and letting $x=(x_{11}, \ldots, x_{1q}, x_{21}, \ldots, x_{2q}, \ldots, x_{pq})$.

Fiduccia[Fid71] proved a theorem similar to Winograd's, but involving submatrices rather than columns:

Theorem 12:

Let Φ, ϕ, x and F be as in Theorem 10. If Φ has a u by v submatrix S such that there are no nontrivial vectors α, and β such that $\alpha S \beta$ is in zero, then at least $u+v-1$ multiplications/divisions are required to compute $\Phi x+\phi$.

Immediate corollaries to this theorem are that at least three real multiplications are required to compute the product of two complex numbers (also proved by Munro[Mu71a]) and that at least seven real multiplications are required to compute the product of two quaternions.

Winograd[Wi70] similarly generalized Motzkin's result (part of Theorem 8) on the number of multiplications when preconditioning is allowed:

Theorem 13:

Let Φ, ϕ, x and F be as in Theorem 10. If there are u column vectors in Φ such that no nontrivial linear combination of them (over F) is in F^m, then any scheme with preconditioning computing $\Phi x+\phi$ requires at least $\lfloor (u+1)/2 \rfloor$ multiplications/divisions.

Motzkin's result follows from this exactly as Pan's followed from Theorem 10, and we have a corollary similar to Theorem 11:

Theorem 14:

Let X and y be as in Theorem 11, then every algorithm for computing Xy requires at least $pq/2$ multiplications/divisions which do not depend only on the entries of X or only on the entries of y.

Winograd[Wi70] showed the possibility of approaching the lower bound given in Theorem 14, by giving an algorithm, which uses preconditioning, to compute Xy in $p\lceil q/2 \rceil + \lfloor q/2 \rfloor$ multiplications; this algorithm then leads to an algorithm to multiply two n by n matrices in $n^2\lceil n/2 \rceil + 2n\lfloor n/2 \rfloor$ or approximately $n^3/2$ multiplications; Winograd's result is somewhat surprising since the usual method of matrix multiplication, that is by the definition, requires n^3 multiplications, and it had not been thought that this could be diminished.

This work was soon followed by an astonishing result of Strassen,[Str69] who showed that two n by n matrices could be multiplied using only $4.7n^{\lg 7}$ (about $4.7n^{2.81}$) arithmetic operations. Strassen's method is based on a clever trick by which 2 by 2 matrices are multiplied using only seven scalar multiplications (instead of eight) and eighteen scalar additions:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} w & x \\ y & z \end{pmatrix}=\begin{pmatrix} (a+d)(w+z)+(b-d)(y+z)+d(y-w)-(a+b)z & (a+b)z+a(x-z) \\ (c+d)w+d(y-w) & (a+d)(w+z)+a(x-z)-(c+d)w+(c-a)(w+x) \end{pmatrix}$$

Since this trick does not make use of commutivity of multiplication, it follows that the method generalizes to higher order matrices by decomposing them into blocks. Strassen goes on to apply his methods to matrix inversion, computing the determinant, and solving linear systems of equations and he shows that each of these can be done in $0(n^{\lg 7})$ arithmetic operations, provided certain submatrices are nonsingular.

It is not, in general, known whether or not Strassen's method is optimal. Hopcroft and Kerr[Hop71a] have worked on this problem and they give a generalization of Strassen's method for multiplying m by 2 times 2 by n matrices which requires $\lceil(3m+1)n/2\rceil$ multiplications. They then show that this number of multiplications is minimal for the cases $n=m=3$ and $m=2$, n arbitrary; the optimality of Strassen's method for 2 by 2 matrices follows immediately from their results.

Several years prior to the work of Strassen and Winograd, Kljuev and Kokovkin-Shcherbak[Kl65] had approached the problem of the solution of an n by n linear system in a different manner. Using a detailed examination of the number and placement of zeroes in the matrix, they proved

Theorem 15:

If only operations on entire rows are permitted then $\frac{1}{6}n(n-1)(2n-1)+n^2$ additions/subtractions and $\frac{1}{3}n(n^2+3n-1)$ multiplications/divisions are required to solve an n by n system of linear equations.

Since these are exactly the numbers of operations required by Gaussian elimination, we have as a corollary that Gaussian elimination is optimal, when one is restricted to operating on entire rows. Strassen's method is faster, but it uses operations on submatrices rather than on rows. In References Kl66, Kl67, Ko68, and Ko70 Kljuev and Kokovkin-Shcherbak extended their previous work and by similar methods established lower bounds for the number of arithmetic operations required for the transformation of matrices.

MISCELLANEOUS PROBLEMS

This section is devoted to a potpourri of results, which stand more or less alone, without a general frame of reference.

Nonlinear equations

Given a nonlinear equation, the problem is to approximate the solution to within a prescribed accuracy using arithmetic operations. Vashakmadze[Vas69] has established some lower bounds on the minimum number of operations necessary to approximate solutions to certain differential equations, and Emel'yanov and Il'in[Em67] studied the same question for certain integral equations.

Various results have been discovered concerning the optimality of iterative root finding methods such as the secant method or Newton's method. For example, Rissanen[Ri71] has shown that the secant method is, in a sense, optimal among all algorithms which use the "same amount of information" and which also satisfy a certain "smoothness" condition. Much work has also been done to find the best starting values for Newton's method applied to square roots; for example, References Mou67, Kin69, and Ster69. Recently, Paterson (personal communication), using the "standard" definition of the efficiency of an iterative scheme, has shown Newton's method to be optimal, for the calculation of square roots (in the sense that no rational scheme can have greater efficiency).

Scalar arithmetic

So far, we have been concerned only with how many operations need to be performed, not considering the time required for individual operations. The usual method for adding/subtracting two n digit numbers requires time proportional to n and the usual method of multiplication requires time proportional to n^2. Can either of these methods be improved upon? For addition/subtraction no substantial improvement is possible since the usual time is about n "cycles," and there are $2n$ inputs (digits) while on each cycle one can use at most two of the inputs. Multiplication, however, can be done more quickly than the usual method. Karatsuba and Ofman[Kara62] developed a method which requires time proportional to $n^{\lg 3} \approx n^{1.57}$. Toom[Too63] generalized this algorithm and proved that for all $\epsilon > 0$ there is a multiplication algorithm such that the time required to multiply two n digit numbers is $0(n^{1+\epsilon})$. Schonhage and Strassen[Scho71] devised a different algorithm which requires at most time $0(n \log n \log \log n)$; this is the most efficient algorithm known, but it has not been proved optimal.

The only non-trivial optimality result is due to Cook and Aanderaa[Co69] where they proved that on a bounded activity machine, an on-line "super" Turing machine, multiplication cannot be performed in less than time $0(n \log n / (\log \log n)^2)$.

Ofman[Of62] introduced the idea of studying the delay time of a circuit and the number of elements in a circuit which computes some given function, say addition. Toom[Too63] extended this work to multiplication and Winograd[Wi65,Wi67] and Spira[Sp69] derived lower bounds on the number of delays and elements required for various operations. Elementary discussions of some of this material appears in References Arb69 and Wi68. This work is not really germane to the present discussion since it deals with circuitry rather than computational schemes of a programmable nature.

Graphs

Other than information theory arguments, there is only one known technique for establishing lower bounds on graph algorithms: if the graph of n nodes is presented as an n by n binary connection matrix then $0(n^2)$ operations are needed to examine all of the arcs of a graph. Based on this observation, Holt and Reingold[Hol72] have shown that determining shortest paths, the existence of cycles, and connectedness requires $0(n^2)$ operations for graphs with n nodes. These results show that the "usual" $0(n^2)$ algorithms for these problems are optimal to within a multiplicative constant factor, when the graph is represented as a connection matrix.

Hopcroft and Tarjan[Hop71b] have devised an $0(n \log n)$ algorithm which determines whether or not an n node graph is planar; Tarjan[Ta71] has improved this to $0(n)$. These algorithms do not violate the lower bound of $0(n^2)$ since a planar graph of n nodes can contain at most $3n-2$ arcs; additionally, their algorithm requires that the graph be presented in a certain linked list format, rather than as a connection matrix.

Computing the transitive closure of a graph is another well-known problem in this area. Warshall[Wa62] gave an $0(n^3)$ algorithm for computing the transitive closure of a graph from its connection matrix. Furman[Fu70] applied Strassen's method of matrix multiplication to that same computation obtaining an $0[n^{\lg 7}(\log n)^{2+\epsilon}]$; Arlazarov, Dinic, Kronrod, and Faradzhev[Arl70] developed another $0(n^3)$ algorithm. Recently, it has been shown[Mu71c,Fis71] that the problems of doing Boolean matrix multiplication and of computing the transitive closure are equivalent to one another; that is, given an $0(f)$ algorithm to multiply Boolean matrices, we can derive an $0(f)$ algorithm for transitive closure, and vice versa. In connection with this result, an $0(n^{\lg 7}(\log n)^{1+\epsilon})$ algorithm is developed.

Harper and Savage (unpublished) have shown that the problem of matching bipartite graphs with n nodes is at least $0(n^3)$ in complexity when the computation is done by combinatorial switching network. In contrast with this, Hopcroft and Karp[Hop71c] have developed an $0(n^2\sqrt{n})$ algorithm, using a different model of computation.

Maximization of unimodal functions

A unimodal function of one variable is function f which has a unique maximum $\bar{x}$; it is characterized by

$$x < y < \bar{x} \Rightarrow f(x) < f(y)$$

$$x > y > \bar{x} \Rightarrow f(x) < f(y).$$

Suppose we want to locate, to within a unit interval, this unique maximum; what is the smallest number of function evaluations required? This question was first studied by Kiefer.[Kie53,Kie57] He showed that if we are to locate the maximum over an interval $[0, L]$ and $F_n \leq L < F_{n+1}$, where F_j is the jth Fibonacci number, then $n+1$ function evaluations are necessary and sufficient. Karp and Miranker[Karp68] characterized optimal strategies when parallel function evaluations are allowed.

REFERENCES

Arb69 M A ARBIB
Theories of abstract automata
Prentice-Hall Inc Englewood Cliffs NJ 1969
section 3.2

Arl70 V L ARLAZAROV E A DINIC
M A KRONROD I A FARADZHEV
On economical construction of the transitive closure of an oriented graph
Dokl Akad Nauk SSSR 194 1970 pp 487-488 (Russian)
English translation in Soviet Math Dokl 11 1970
pp 1209-1210

Be58 E G BELAGA
Some problems involved in the calculation of polynomials
Dokl Akad Nauk SSSR 123 1958 pp 775-777 (Russian)
See Math Reviews 21 1960 review number 3935

Be61 _____
On computing polynomials in one variable with initial conditioning of the coefficients
Problemy Kibernet 5 1961 7-15 (Russian) English translation in Problems of Cybernetics 5 1964 pp 1-13

Bo71 A BORODIN
Horners rule is uniquely optimal
Theory of Machines and Computations
Edited by Z Kohavi and A Paz
Academic Press New York 1971 pp 45-58
Computations Haifa Israel 1971

Br39 A BRAUER
On addition chains
Bull Amer Math Soc 45 1939 pp 736-739

Co69 S A COOK S O AANDERAA
On the minimum computation time of functions
Trans Amer Math Soc 142 1969 pp 219-314

Em67 K V EMEL'YANOV A M IL'IN
Number of arithmetical operations necessary for the approximate solution of Fredholm integral equations of the second kind
Zh Vychisl Mat i Mat Fiz 7 (1967) pp 905-910
(Russian) English translation in USSR Comput Math and Math Phys 7 1967 pp 259-266

Ev64 J EVE
The evaluation of polynomials
Numer Math 6 1964 pp 17-21

Fid71 C M FIDUCCIA
Fast matrix multiplication
Proc of Third Annual ACM Symp on Theory of Computing 1971 pp 45-49

Fis71 M J FISCHER A R MEYER
Boolean matrix multiplication and transitive closure
IEEE Conf Record of the Twelfth Annual Symps on Switching and Automata Theory 1971 pp 129-131

Fo59 L R FORD JR S M JOHNSON
A tournament problem
Amer Math Monthly 66 1959 pp 387-389

Fu70 M E FURMAN
Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph
Dokl Akad Nauk SSSR 194 1970 p 524 (Russian)
English translation in Soviet Math Dokl 11 1970 p 1252

G156 A GLEASON
Unpublished internal IBM memorandum 1956

Gr71 R L GRAHAM
Sorting by comparisons
Computers in Number Theory edited by A O L Atkin and B J Birch Academic Press New York 1971 pp 263-269

Ha69a A HADIAN
Optimality properties of various procedures for ranking n different numbers using only binary comparisons
Technical Report Number 117 Department of Statistics University Minnesota Minneapolis Minnesota 1969

Ha69b _____ M SOBEL
Selecting the t^{th} largest using binary errorless comparisons
Colloquia Mathematica Societatis Janoś Bolyai
Balatonfüred Hungary 1969 pp 585-589

Ha70 _____ _____
Ordering the t largest of n items using binary comparisons
Proc Second Chapel Hill Conf on Combinatorial Math and its Application 1970

Ho170 R C HOLT E M REINGOLD
On the time required to detect cycles and connectivity in directed graph
To appear in Math Systems Theory 1972

Hop71a J HOPCROFT L KERR
On minimizing the number of multiplications necessary for matrix multiplication
SIAM J Appl Math 20 1971 pp 30-36

Hop71b _____ R TARJAN
Planarity testing in VlogV steps
Information Processing 1971 Proc of IFIP Congress 71 1971

Hop71c _____ R M KARP
A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs
IEEE Conf Record of the Twelfth Annual Symps on Switching and Automata Theory 1971 pp 32-34

Hw71 F K HWANG S LIN
Optimal merging of 2 elements with n elements
Acta Informatica 1 1971 pp 145-158

Kara62 A KARATSUBA YU OFMAN
Multiplication of multidigit numbers on automata
Dokl Akad Nauk SSSR 145 1962 293-294 (Russian)
English translation in Soviet Physics Dokl 7 1963 pp 595-596

Karp68 R M KARP W L MIRANKER
Parallel search for a maximum
J Combinatorial Theory 4 1968 pp 19-35

Kie53 J KIEFER
Sequential minimax search for a maximum
Proc Amer Math Soc 4 1953 pp 502-507

Kie57 _____
Optimum sequential search and approximation methods under minimum regularity assumptions
J Soc Indust Appl Math 5 1957 pp 105-137

Kin69 R F KING D L PHILLIPS
The logarithmic error and Newton's method for the square root
Comm ACM 12 1969 pp 87-88

Kis62 S S KISLICYN
On a bound for the least mean number of pairwise comparisons needed for a complete ordering of n objects with different weights
Vestnik Leningrad Univ Ser Mat Meh Astronom 17 1962 pp 162-163 (Russian with English summary)

Kis63 _____
An improved bound for the least mean number of comparisons needed for the complete ordering of a finite set
Vestnik Leningrad Univ Ser Mat Meh Astronom 18 1963 pp 143-145 (Russian with English summary)
English translation in Document Number AD668523 available from the Clearinghouse Department of Commerce (this is an edited machine translation by the USAF 1967) pp 178-181

Kis64 ______
On the selection of the k^{th} element of an ordered set by pairwise comparisons
Sibirsk Mat Zh 5 1964 pp 557-564 (Russian) See Math Reviews 29 1965 review number 2198

Kl65 V V KLJUEV N I KOKOVKIN-SHCHERBAK
On the minimization of the number of arithmetic operations for the solution of linear algebraic systems of equations
Zh Vychisl Mat i Mat Fiz 5 1965 pp 21-23 (Russian) English translation in USSR Comput Math and Math Phys 5 1965 pp 25-43. Also translated in Technical Report CS24 Department of Computer Science Stanford University June 1965

Kl66 ______ ______
Minimization of the number of arithmetical operations in a transformation of matrices
Ukrain Mat Zh 18 1966 pp 122-128 (Russian) See Math Reviews 34 1967 review number 3767

Kl67 ______ ______
Minimization of computational algorithms in certain transformations of matrices
Zh Vychisl Mat i Mat Fiz 7 1967 pp 3-13 (Russian) English translation in USSR Comput Math and Math Phys 7 1967 pp 1-14

Kn62 D E KNUTH
Evaluation of polynomials by computers
Comm ACM 6 1962 pp 595-599

Kn69 ______
The art of computer programming, Volume 2
Addison-Wesley Reading Massachusetts 1969 sections 4.3.3 4.6.3 and 4.6.4

Kn72 ______
The art of computer programming, Volume 3
Addison-Wesley Reading Massachusetts to appear 1972 section 5.3

Ko68 N I KOKOVKIN-SHCHERBAK
Minimization of computational algorithms in the solution of the elimination problem
Zh Vychisl Mat i Mat Fiz 8 1968 pp 1096-1101 (Russian) English translation in USSR Comput Math and Math Phys 8 1968 pp 212-219

Ko70 ______
On minimization of calculation algorithms in solutions of arbitrary systems of linear equations
Ukrain Mat Zh 22 1970 pp 494-502 (Russian)

Ma71 K MARUYAMA
Parallel methods and bounds of evaluating polynomials
Report Number 437 Department of Computer Science University of Illinois Urbana Illinois 1971

Mot55a T S MOTZKIN
Evaluation of polynomials
Bull Amer Math Soc 61 1955 p 163 abstract only

Mot55b ______
Evaluation of rational functions
Bull Amer Math Soc 61 1955 p 163 abstract only

Mou67 D G MOURSUND
Optimal starting values for Newton-Raphson calculation of $x^{1/2}$
Comm ACM 10 1967 pp 430-432

Mu71a I MUNRO
Some results concerning efficient and optimal algorithms
Proc of Third Annual ACM Symp on Theory of Computing 1971 pp 40-44

Mu71b ______ M S PATERSON
Optimal algorithms for parallel polynomial evaluation
IEEE Conf Record of the Twelfth Annual Symps on Switching and Automata Theory 1971 pp 132-139

Mu71c ______
Efficient determination of the transitive closure of a directed graph
Information Processing Letters 1 1971 pp 56-58

Of62 YU OFMAN
On the algorithmic complexity of discrete functions
Dokl Akad Nauk SSSR 145 1962 pp 48-51 (Russian). English translation in Soviet Physics Dokl 7 1963 pp 589-591

Os54 A M OSTROWSKI
On two problems in abstract algebra connected with Horner's rule
Studies in Mathematics and Mechanics Presented to R von Mises Academic Press New York 1954 pp 40-48

Pan59 V YA PAN
Schemes for computing polynomials with real coefficients
Dokl Akad Nauk SSSR 127 1959 pp 266-269 (Russian). See Math Reviews 23 1962 review number B560. French translation available from Laboratoire Central, De L'Armement Traductions 16 bis Avenue Prieur-de-la-Cote-d'Or 94 Arcueit France; translation number T-1.524

Pan61 ______
Certain schemes for the evaluation of polynomials with real coefficients
Problemy Kibernet 5 1961 pp 17-29 (Russian). English translation in Problems of Cybernetics 1964 pp 14-32

Pan62 ______
On several ways of computing values of polynomials
Problemy Kibernet 7 1962 pp 21-30 (Russian). See Comput Rev 3 1962 review number 3329

Pan65 ______
The computation of polynomials of fifth and seventh degree with real coefficients
Zh Vychisl Mat i Mat Fiz 5 1965 pp 116-118 (Russian). English translation in USSR Comput Math and Math Phys 5 1965 pp 159-1611

Pan66 ______
Methods of computing values of polynomials
Uspehi Mat Nauk 21 1966 pp 103-134 (Russian). English translation in Russian Math Surveys 21 1966 pp 105-136

Pat71 M S PATERSON L STOCKMEYER
Bounds on the evaluation time for rational polynomials
IEEE Conf Record of the Twelfth Annual Symps on Switching and Automata Theory 1971 pp 140-143

Po69 I POHL
A minimum storage algorithm for computing the median
Report number RC2701 IBM Yorktown Heights New York 1969

Ra71 M O RABIN
Proving simultaneous positivity of linear forms
Third Annual ACM Symp on Theory of Computing 1971 Invited address

Ri71 J RISSANEN
On optimal root-finding algorithms
J Math Anal Appl 36 1971 pp 220-225

Sa61 M SANDELIUS
On an optimal search procedure
Amer Math Monthly 68 1961 pp 133-134

Scho71 A SCHONHAGE V STRASSEN
Fast multiplication of large numbers
Computing 7 1971 pp 281-292 (German with English summary)

Schr32 J SCHREIER
On tournament eliminations systems
Mathesis Polska 7 1932 pp 154-160 (Polish)

Sl51 J SLUPECKI
On the system S of tournaments
Colloq Math 2 1951 pp 286-290

Sm47 C A B SMITH
The counterfeit coin problem
Math Gaz 31 1947 pp 31-39

Sp69 P M SPIRA
The time required for group multiplication
J Assoc Comput Mach 16 1969 pp 235-243

Stei50 H STEINHAUS
Mathematical snapshots
Oxford University Press New York first edition 1950 second edition 1960

Stei58 ———
Some remarks about tournaments
Calcutta Mathematical Society Golden Jubilee Commemoration Vol 1958-1959 Part II pp 323-327

Ster69 P H STERBENZ C T FIKE
Optimal starting approximations for Newton's method
Math Comp 23 1969 pp 313-318

Str69 V STRASSEN
Gaussian elimination is not optimal
Numer Math 13 1969 pp 354-356

Ta71 R TARJAN
Ph D Thesis Stanford 1971

Tod55 J TODD
Motivation for working in numerical analysis
Comm Pure Appl Math 8 1955 pp 97-116

Too63 A L TOOM
The complexity of a scheme of functional elements realizing the multiplication of integers
Dokl Akad Nauk SSSR 150 1963 pp 496-498 (Russian). English translation in Soviet Math Dokl 4 1963 pp 714-716

Val59 R E VAL'SKII
The smallest number of multiplications necessary to raise a number to a given power
Problemy Kibernet 2 1959 pp 73-74 (Russian). English translation in Problems of Cybernetics 2 1961 pp 395-397

Van70 M H VAN EMDEN
Increasing the efficiency of quicksort
Comm ACM 13 1970 pp 563-567

Vas69 T S VASHAKMADZE
On some optimal algorithms
Thbilis Sahelmc Univ Gamoqeneb Math Inst Srom 1 1969 pp 7-16 (Russian with Georgian summary)

Wa62 S WARSHALL
A theorem on Boolean matrices
J Assoc Comput Mach 9 1962 pp 11-12

We65 M B WELLS
Applications of a language for computing in combinatorics
Information Processing 1965 Proc of IFIP Congress 65 1965 pp 497-498

Wi65 S WINOGRAD
On the time required to perform addition
J Assoc Comput Mach 12 1965 pp 277-285

Wi67 ———
On the time required to perform multiplication
J Assoc Comput Mach 14 1967 pp 793-802

Wi68 ———
How fast can computers add?
Scientific American October 1968 pp 93-100

Wi70 ———
On the number of multiplications necessary to compute certain functions
Comm Pure Appl Math 23 1970 pp 165-179

Analysis of combinatory algorithms— A sample of current methodology

by W. D. FRAZER

IBM T. J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

The study of the computational efficiency and inherent limitations of algorithms for computer solution of problems drawn from classical continuous mathematics has been with us as long as general purpose computers themselves. Similar studies of algorithms of other kinds, however, have been sporadic and isolated until fairly recently. With the growing realization of the critical role played by combinatory algorithms in any instance in which the computer is employed as a logical decision maker, however, has come intensive and widespread interest in understanding such algorithms. Further, these studies have progressed to the point where patterns in analysis have begun to emerge and a summary seems in order. We shall focus our attention here on analytic approaches and methodology, rather than on the specifics of the results obtained. For a summary of the latter, the reader is referred to a companion paper elsewhere in this volume.[16]

To begin with, one might consider what constitutes a combinatory algorithm and what does not, but rather than restrict the nature of the analyses we shall consider, let us say merely that we shall not discuss problems in numerical analysis, in the theory of automata, or in arithmetic and leave it at that. We shall take it as our primary concern to try to summarize the kinds of information people have attempted to learn about various algorithms, and then indicate some of the tools and techniques they have employed. An effort will be made to point out common threads where possible, and to indicate potential directions for future development where this seems appropriate. As illustrative vehicles for the discussion we shall focus on some simple, but basic, examples. To the best of our knowledge, the analyses and proofs (as distinct from the results) are new except where noted.

Broadly speaking, analyses of combinatory algorithms can be classified in two "dimensions," viz.,

	Analysis of a Single Algorithm	Analysis of a Class of Algorithms
Analysis of a Single Case (usually worst or best)		
Analysis of a Probability Distribution of Cases		

The kinds of information sought through analysis are typically: (a) Does the algorithm(s) do what is desired? (b) How does it (or they) perform if all goes favorably or unfavorably? (c) How good is performance "on the average"?, and (much less frequently) (d) How average is average—i.e., what is the variance?

Of course, the study of algorithm performance presupposes a choice of a figure of merit for that performance. This choice is frequently quite difficult to make in a manner which reflects adequately the amount of real computation involved. In the case of algorithms for sorting, for example, a common figure of merit is the number of comparisons made. This measure is a valid indicator of actual program performance, however, only if comparisons constitute the bulk of the computation. An algorithm which performs address calculations based on record key values may perform few or even no compares; another, which does base its course of action on the outcomes of compares, may nevertheless access memory often to move items around without comparing them, and thus be greatly inferior in reality to one which makes a few more comparisons. It is primarily because of a lack of reasonably close correspondence with reality in the underlying models and measures of performance that almost all of the work on "computational complexity" in the literature of automata theory finds little or no use for our purposes.

In the course of our examination of the techniques of analysis of algorithms, we will observe repeatedly that there are two qualitative "depths" to which such an analysis is normally pursued. The first is a general level where all that is specified is the operations performed by the algorithm and the general form of the data structures it accesses. The second is the level of at least a source language implementation in which considerable attention must be paid to the implementation of these operations and data structures. An operation such as "find a circuit" of a linear graph, for example, is far more complex than "compare this item with that item," even in cases where the latter requires reference to a lexicography. Again, the data structures or accessing patterns required by the general specification of an algorithm may not be directly implementable or may incur excessive overhead in implementation.

With these caveats in mind, let us begin an examination of some of the main themes of the extant literature in the analysis of algorithms.

CERTIFICATION: DOES THE ALGORITHM DO WHAT IS DESIRED?

Many combinatory algorithms, generally stated, do not require detailed formal certification of correctness and/or termination because it is obvious that these properties are satisfied. Most sorting algorithms, for example, fall into this category, as do most search algorithms. As noted earlier, there are two levels at which one can question these properties—that of the abstract or "pure" statement of the algorithm, and that of a particular implementation. Obviously, even verification at the level of implementation in a source language program will not guarantee that the compiled program will perform as desired, but it is a much better guarantee than verification at the abstract level. In practice, however, both levels are frequently desirable, since the techniques required for one are often qualitatively different from those for the other, and it is most desirable to know that the algorithm is free of flaws before carrying it to the level of a detailed implementation. As an example of some of these ideas let us consider the following problem:

Consider a linear graph consisting of a finite collection of vertices, some of which are joined pairwise by a collection of edges. With each such edge is associated a positive length, and each pair of vertices may be joined by at most one edge. The only other restriction is that there must be some sequence of edges, or path, joining any pair of vertices (i.e., no group of vertices is "isolated"). A (spanning) *tree*, T, is a collection of edges having the property that within T there is exactly one path joining any pair of vertices. Our problem is to find a tree of minimum total length, $T_{\min}$.

A very effective algorithm (in fact, the best published to date) for this problem is due independently to Prim[15] and Dijkstra:[3] We begin by selecting an arbitrary vertex, v_0, and assigning to $T_{\min}$ the shortest of its incident edges, say one linking v_0 to v_1, denoted $\langle v_0, v_1\rangle$. Next, from all of the edges linking either v_0 or v_1 to other vertices we select the shortest, which links v_0 or v_1 to, say, v_2. We now repeat the process, looking this time for the shortest edge joining v_0, v_1, or v_2 to some other vertex, and continue in this manner until all vertices have been included. Notice that at any time we need only keep track of a shortest edge from some vertex already in $T_{\min}$ to each of those vertices not yet included. As each new vertex is joined to $T_{\min}$, the lengths of its edges to vertices not yet in $T_{\min}$ are compared with the current minima, and the latter replaced where appropriate.

That this procedure will terminate is clear, even from the informal statement just given, because a new vertex is added to the tree at each step. It is only slightly less clear that the resulting collection of edges will be a tree, but a moment's reflection will convince the reader that a path between any pair of vertices must be found and will be unique. There is no way to add to $T_{\min}$ an edge joining vertices already joined by a path, since the only edges under consideration are those joining such vertices to vertices not yet reachable by edges in $T_{\min}$.

The remaining major question, that of the minimality of $T_{\min}$, is less obvious. Suppose $T_{\min}$ were not minimal, and suppose there were some minimal tree T' which differed from $T_{\min}$ in a minimal number of edges. Let E be the set of edges of $T_{\min}$ not in T' and let e be a shortest edge of E. Now, in $T_{\min}$, e uniquely joins the set of vertices already in $T_{\min}$ at the time e was added, V_e, to the set of the remaining vertices of the graph, W_e. Since T' is a tree, it also has at least one edge

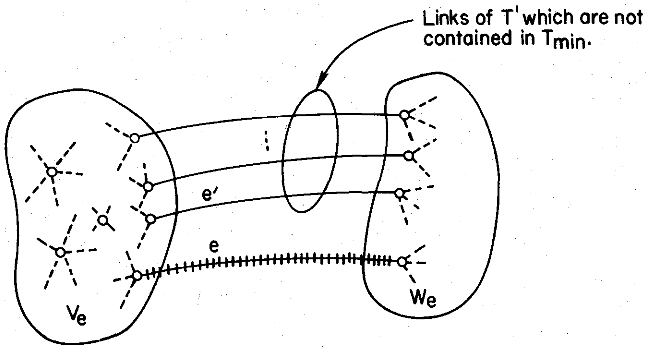


Figure 1

joining V_e to W_e; furthermore, adding e to T' will create two paths (in $T' \cup \{e\}$) between some vertices of V_e and some of W_e, one path via e and one via some edge, e', of T' (cf. Figure 1). Now length $(e') \geq$ length (e), since e was chosen by our algorithm. On the other hand, if we substitute e for e' in T' the result will still be a tree, T_e'. (The vertices in V_e and W_e formerly connected in T' via e' will now be connected via e.) But T_e' can have no less total length than T', since T' was minimal; hence, length $(e) \geq$ length (e'). Combining these observations, length $(e) =$ length (e') and length $(T_e') =$ length (T'). But T_e' differs from $T_{\min}$ in one fewer edges than T' and this contradicts our choice of T'; thus, T' does not exist and $T_{\min}$ is minimal, as required.

We note in passing that the device employed here is a powerful tool in the analysis of graph algorithms. One verifies the optimality of the solution produced by an algorithm by postulating the existence of an optimal solution differing from the generated solution in a minimal way, and then constructively refutes the minimality of this difference.

Having satisfied ourselves as to the basic soundness of the algorithm, the next step is to embody it in a program and verify the correctness of the program. We will not pursue this step in detail here for lack of time and space but will rely upon the reader's experience to assist in understanding the qualitative difference between such an undertaking and what we have just discussed. To begin with, even in a higher level language one is now faced with a host of syntax related problems having to do with whether variables are of the appropriate type and with the legitimacy of program statements, for example. There are also semantic questions such as whether loop indices are tested against range boundaries before or after incrementing and whether their values are preserved intact where required, as well as implementation-dependent considerations such as whether input and output is performed correctly, and so on. Next come considerations of program logic and control flow including again the question of termination, this time for the program. Finally, and perhaps most difficult, one has the question of whether the program actually embodies the algorithm at all. This process is clearly vastly more detailed than our proof above, even if the latter were to be thoroughly formalized. The most commonly accepted current approach to such a certification entails first supplementing the program code with assertions about statements in that code and verifying by hand that these assertions reflect accurately the desired properties of the statement [cf. e.g., References 5, 10, 18]. Next the assertions, couched in a form more suitable for treatment via classical mathematical proof techniques, are themselves verified. There is substantial effort under way in a number of areas to address these problems. It is abundantly clear that the first process—that of establishing the correspondence between statements and assertions—could benefit from mechanical assistance. On the other hand, a hopeful sign is that the prospects for providing such assistance appear reasonably good as a consequence of increased understanding of programs resulting from, for example, compiler optimization studies.[1] Automatic verification of the assertions is considerably more difficult, though a good deal of work is being done [cf. References 5, 10, 18]. There are a number of recent instances, however, in which humans have been quite successful in accomplishing this without undue tedium, and thus automatic *assistance* may be all that is required for a viable system.

EXTREME PERFORMANCE: WHAT HAPPENS AT BEST (OR AT WORST)?

Granted that one has assured himself of the validity of an algorithm, it is next natural to ask how well the algorithm performs its assigned task. Here again, we find the opportunity (or need) for two levels of analysis: one at the general algorithm level and the other at the detailed program level.

In the case of our example, the best and worst cases would arise when the graph being processed consisted of a single path and a "complete graph" (i.e., when each vertex has an edge link to every other vertex), respectively. It is clear that a reasonable figure of merit for the performance of this algorithm must include some measure of the work performed in finding the shortest edge at each stage; it may or may not be appropriate in addition to examine the amount of storage space required by the procedure.

Suppose that we have n vertices. Then in the former case, only $(n-1)$ edges need be examined since only a single choice is available at each stage, while in the latter one must choose the first edge from among $(n-1)$, the second from among $2\times(n-2)$, etc., and the last from among $(n-1)$ again. It should be fairly obvious that one can find the smallest of k objects in $(k-1)$ comparison steps, and that one can do no better than this. (This is an instance in which one can make a very strong statement about existence and optimal performance of a whole family of algorithms.) Thus, a first analysis might lead to the conclusion that $\sum_{i=1}^{n-1} i(n-i) = [n(n-1)(n+1)/6]$ comparison steps are required. Recall, however, that as each new vertex joins the growing $T_{\min}$, its edges to vertices not yet included need be compared only with the *shortest* of the corresponding edges from vertices already included, and not with all such edges. This is the essence of the efficiency of the procedure. Thus $(n-1)$ comparison steps are required for selecting v_1, followed by $(n-2)$ steps to establish a set of $(n-2)$ minimal length edges to other vertices and then $(n-2)$ steps to find a shortest such edge in order to pick v_2. The total number of comparison steps is thus $(n-1)+2(n-2)+2(n-3)+\cdots+2(2)+1=n(n-2)$. At a cost of some additional storage to keep track of the current minima, we have avoided a cumbersome search at each stage and improved performance from $0(n^3)$ to $0(n^2)$. This serves to emphasize that even at this general level of analysis, one must exercise great care to be sure that he has captured the essence of the algorithm, for $0(n^2)$ performance is significantly different from $0(n^3)$.

Of course, the analysis is not yet complete. We must still account for the work required to keep track of which vertices and edges are already in $T_{\min}$ and which are not, and to do the bookkeeping for the maintenance of the list of minimal distances, for example. An accurate account of this kind, however, really requires that one move to the next level of analysis—that of the program itself.

Not surprisingly, the level of detail in such an analysis is again considerable. Still, the number of parameters is not overwhelming, for the flow of control in a program—at least in a *good* one—is normally highly systematic. In addition, the certification process has by this time assured one that his program is both consistent and free of infinite loops. Thus, each loop will be entered and exited an equal number of times, and this number can be used as a parameter characterizing the number of executions of a whole set of statements within and without the loop; also, several of these parameters are often available from the analysis of the algorithm at the "general" level. Again, we shall omit this phase of the analysis in order to move on. It is worthwhile noting, however, that the *kind* of analysis would be essentially similar to that which we have just seen—i.e., finite discrete mathematics involving integer quantities—although quite likely more complex than our simple example [cf. Reference 13].

There is a drawback in this kind of extreme analysis, however, which may not yet be apparent. Observe that there is a major qualitative difference between the best and worst case performance. It is probably fairly unlikely that one would encounter either in practice, and thus, while the time performance for most graphs lies somewhere between these extremes, the range is so large as to furnish little information. This is the motivation for undertaking the kind of analyses we shall consider next, those which attempt to quantify "average" performance. Before moving on to consider analysis of average performance, however, let us digress briefly to consider a problem in which extreme performance is of paramount importance.

Consider a two-input, two-output device which compares the values of, and then propagates, its inputs in such a way that the larger input reappears on the first output line and the smaller on the second. The objective in the construction of "sorting networks"[2] is to devise arrays of such devices which accept a string of input items as input and then shuttle these input items about internally over fixed connections to permute the items into sorted order. For such networks one figure of merit is obviously the number of comparisons (i.e., devices) required. The major difference between this approach and most standard approaches to sorting, however, is that here the set of comparisons is immutable regardless of the input sequence (since the connections are fixed). In return for this inflexibility, one gains a potential for parallel operation and a saving in time. Thus, an equally important figure of merit for most applications is the total *time* required to sort, and this is determined by the *maximum* number of comparisons made against any item in moving from input to output. In this instance, as with many involving algorithms whose course of action is essentially independent of the values or positions of data items, extreme performance is of primary interest.

EXPECTED PERFORMANCE: HOW GOOD IS THE ALGORITHM "ON THE AVERAGE"?

A more difficult kind of analysis of algorithm performance, in general, is that required to determine expected or "average" performance. Still, it is precisely this kind of analysis which usually yields the most insight into actual observed algorithm performance. Outside the realm of information theory where random coding arguments have found some utility, it is highly unusual to find an analysis of expected performance of families of algorithms; far more often is a single algorithm analyzed in this way, and this is the kind of analysis we shall pursue next. For reasons to which we shall return later, we shall forsake our minimal spanning tree algorithm and concentrate on another algorithm to illustrate how such an analysis is carried out.

The algorithm we shall examine is one which sorts (i.e., permutes) an input sequence of records, each of which consists of both a "key" chosen from some linearly ordered set and other associated information, into, say, ascending lexicographic key order. For convenience in analysis and description, we shall assume that all of the key values are distinct. That this makes no essential difference will be easily verifiable in retrospect. For brevity, we shall refer to "record values" when speaking of the values of keys associated with these records. The algorithm is known as "Quicksort"[8] and it works as follows.

Quicksort constructs a binary tree whose vertices correspond to the, say n, records in the input sequence. The construction of the tree is controlled by the following rule, applied recursively to each subtree: if a record with (key) value x_0 is assigned to vertex i during construction, then any subsequent record, with value x, assigned to the (sub)tree rooted at i, will be assigned to the left subtree of i if $x<x_0$ and to the right subtree if $x>x_0$. Thus, the first input record becomes the root of the Quicksort tree, the second input record the root of one of the two principal subtrees, and so on.

Evaluation of expected values presupposes the existence of a probability distribution over input permutations; we shall adopt the hypothesis that all such permutations are equally likely. Although there are definitely situations in which such is not the case, there are many others to which it is a reasonable approximation. In addition, the results of such an analysis usually are observed to be good indicators of the measured performance of running programs. Parenthetically, although this may seem a strong assumption, it is actually weaker than assuming, for example, that the key values are independent, identically distributed, random variables.

To proceed with the analysis, we need once again to choose a figure of merit, but before we can choose one we need more information about how the algorithm is to be implemented. Suppose the input sequence is in locations numbered 1 through n: Transfer the first record to another location t, and begin a search of $2, 3, \ldots,$ looking for the first record, say x_j in location j, larger than $x_1 (x_j>x_1)$. Following examination, each such record is returned to a location numbered one less than its original one. Now begin a search of locations $n, n-1, \ldots,$ looking for the first record, say x_k in location k, smaller than x_1, and returning each record to its original location. Interchange the contents of locations $(j-1)$ and k—i.e., x_j and x_k. Now all records in locations 1 through $j-1$ are smaller than x_1 and all those in locations k through n are larger; location j is redundant. We now repeat the above process on locations $(j+1)$ through $k-1$ and continue this until the pointers j and k converge to adjacent locations, say, r and $r+1$. Then x_1 is inserted into r, its correct position in the output sequence, and the two sequences in locations 1 through $(r-1)$ and $(r+1)$ through n are each treated by the above process.

At the level of detail we are considering, the only obvious figure of merit is the number of comparisons made in the searches. There are no other obvious memory accesses or operations, except those required to maintain pointers to the boundaries of partitioned subsequences, such as $(r-1)$ and $(r+1)$ above. In fact, it is possible to hold this requirement to quite reasonable limits[6,7,8] and we shall therefore concentrate on the expected number of comparisons, which corresponds to the expected path length of the Quicksort tree.

Let $E(n)$ be the expected compares required to sort a sequence of n records. Then, if $p(r)$ = probability that the root of the Quicksort tree ranks rth in the input sequence,

$$E(n)=\sum_{r=1}^{n} p(r)[E(r-1\mid r)+E(n-r+1\mid r)+(n-1)]$$

where $E(y\mid r)$ denotes the expected compares to sort a sequence of y records conditioned on the fact that r is the root. But under our assumptions, any record is equally likely to be the root; thus $p(r)=1/n$ for all r; furthermore $E(y\ r)$ is similarly independent of r. Thus:

$$E(n)=(n-1)+\frac{2}{n}\sum_{r=1}^{n} E(r-1)$$

Similarly, following Windley:[19]

$$E(n-1)=(n-2)+\frac{2}{(n-1)}\sum_{r=1}^{n-1} E(r-1)$$

$$nE(n)-(n-1)E(n-1)=2(n-1)+2E(n-1) \qquad (1)$$

$$\frac{E(n)}{n+1}-\frac{E(n-1)}{n}=\frac{2(n-1)}{n(n+1)}=\frac{4}{n+1}-\frac{2}{n}$$

$$\frac{E(n)}{n+1}-\frac{E(1)}{2}=\left[\frac{4}{n+1}-\frac{2}{n}+\frac{4}{n}-\frac{2}{n-1}+\cdots+\frac{4}{3}-\frac{2}{2}\right]$$

$$=\frac{4}{n+1}+\sum_{i=2}^{n-1}\frac{2}{i+1}-1$$

or

$$E(n)=2(n+1)\sum_{i=1}^{n}\frac{1}{i+1}-2n$$

$$\approx 1.39(n+1)\log_2 n-0(n)$$

This kind of analysis is roughly the same as that which we encountered previously, and the technique of arranging for some linear combination of the variables on the left which will be equal to a tractable sum is one frequently employed. Often, however, recurrence equations will not succumb to such analysis. Consequently, another technique frequently called upon is the use of generating functions. A *generating function* for a sequence is an infinite formal power series expansion in which the coefficient of x^n is the nth term in the sequence. Thus, if $G(x)$ is the generating function for $E(n)$:

$$G(x)=E(1)x+E(2)x^2+\cdots+E(n)x^n+\cdots$$

Furthermore the generating function for $nE(n)$ is $x(d/dx)G(x)=E(1)x+2E(2)x^2+\cdots$. Now using this fact the recurrence equation (1) can be rewritten in terms of generating functions as

$$x\frac{d}{dx}G(x)=G(2(n-1))+\frac{d}{dx}(x^2G(x))$$

where

$$G(2(n-1))=\frac{2x^2}{(1-x)^2}$$

$$=2x^2+4x^3+6x^4+\cdots$$

Notice that this equation is in fact an equation involving infinite power series and thus constitutes an infinite set of coefficient equations, one for each power of x since the powers of x are linearly independent. Setting $y=G(x)$, we have

$$xy'=[2x^2/(1-x)^2]+2xy+x^2y'$$

$$y'-[2/(1-x)]y=2x/(1-x)^3$$

This differential equation has an integrating factor

$$\exp\left[-\int\left(\frac{2}{1-x}\right)dx\right]=(1-x)^2.$$

Thus

$$(1-x)^2y'-2(1-x)y=2x/(1-x)$$

Integrating both sides:

$$y(1-x)^2=2(-x-\ln(1-x))+c$$

or

$$G(x)=y=\frac{2}{(1-x)^2}\ln\left(\frac{1}{1-x}\right)+\frac{c}{(1-x)^2}-\frac{2x}{(1-x)^2}$$

Now

$$\frac{1}{(1-x)}\ln\left(\frac{1}{1-x}\right)=x+(1+\tfrac{1}{2})x^2+(1+\tfrac{1}{2}+\tfrac{1}{3})x^3+\cdots$$

Thus

$$\frac{d}{dx}\left[\frac{1}{(1-x)}\ln\left(\frac{1}{1-x}\right)\right]$$

$$=1+2(1+\tfrac{1}{2})x+3(1+\tfrac{1}{2}+\tfrac{1}{3})x^2+\cdots$$

$$=\frac{1}{(1-x)^2}\ln\left(\frac{1}{1-x}\right)+\frac{1}{(1-x)^2}$$

$$\therefore y=2\frac{d}{dx}\left[\frac{1}{(1-x)}\ln\left(\frac{1}{1-x}\right)\right]-\frac{2x}{(1-x)^2}+\frac{c-2}{(1-x)^2}$$

Looking at $E(1)$, the coefficient of x in $G(x)$:

$$E(1)=0=4(1+\tfrac{1}{2})-2+2(c-2)$$

$$\therefore c=0$$

and

$$E(n)=2(n+1)\sum_{i=1}^{n+1}\frac{1}{i}-2(n+1)-2n$$

$$=2(n+1)\sum_{i=1}^{n}\frac{1}{i+1}-2n$$

as before.

Although in this instance the generating function solution is somewhat more elaborate than the direct solution, we took the trouble to illustrate both because the existence of a direct solution is rather a fortuitous circumstance. (The example was chosen, in fact, because both kinds of solution were possible.) The availability of all of the functional power of infinite series vastly increases one's flexibility when dealing with problems of analysis, as may be apparent from the operations above. Of course, there are a large number of other approaches to the solution of recurrence and difference equations [cf. References 9, 12, 14, 17].

A final remark is in order regarding the question of expected performance of graph algorithms. The unfortunate fact is that there seems at present to be no way effectively to characterize a "random" graph. The kinds of graphs submitted to graph algorithms normally have the imprint of human intelligence in design or at least selection. This imprint is not captured effectively by choosing a random incidence matrix, for example, nor by any of the other obvious choices of a definition of randomness to which one is tempted. The discovery of a natural and analytically tractable characterization of a random graph remains a major open question in the analysis of algorithms.

VARIANCE: HOW OFTEN DOES "AVERAGE" PERFORMANCE OCCUR?

Expected performance is most probable performance, but for some algorithms actual performance can be a highly variable function of input data as we have noted. Among all of the kinds of analysis to which algorithms are subjected, analysis of the variance of performance is unquestionably the most difficult—and the most often omitted.

There are, however, sometimes legitimate ways in which to justify omission of computation of the variance. For example, in the case of Quicksort, the extremes of performance occur when the tree is a single path (i.e., completely unbalanced) and when it is completely balanced. In the former case, this number of comparisons is $[n(n-1)/2]$, while in the latter it is $(n+1)\log_2(n+1)-2n$. Thus, expected performance is qualitatively the same as best performance, i.e., $0[(n)\log(n)]$. Therefore, even though performance in the worst case can deteriorate somewhat severely, this must be a consequence of a long "tail" in the distribution and hence quite unlikely under the hypothesis of equally likely input permutations.

As a final example, consider the problem of inserting a new record into a sorted sequence of records. Rather than adopt the standard "binary search" approach, however, let us proceed as follows: Given an nth record to be inserted into a sorted sequence of $(n-1)$ records, we shall pick a point, i, *at random*, make a comparison with the ith record, and thereby determine whether the new record belongs among the first $(i-1)$ or the last $(n-i+1)$. (We shall again assume that all key values are distinct.) We will then repeat the "random probe" process on the appropriate subsequence and continue recursively in this way until the proper position for the new record has been found. Again, let us assume that the new record is equally likely to belong anywhere in the sequence—including at the ends. Let $p(k, n) =$ prob. that k compares are required to insert an item into a set of n. Then

$$p(k,n) = \sum_{i=1}^{n}\left[\left(\frac{1}{n}\right)\left(\frac{i}{n+1}\right)p(k-1,i-1)\right.$$
$$\left. + \left(\frac{n-i+1}{n+1}\right)\left(\frac{1}{n}\right)p(k-1,n-i)\right]$$

where

$1/n =$ prob. that the ith record is chosen at random

$i/(n+1) =$ prob. that the new record belongs among the first $(i-1)$, given that i was chosen

$(n-i+1)/(n+1) =$ prob. that the new record belongs among the last $(n-i)$, given that i was chosen

By symmetry

$$p(k,n) = \frac{2}{n(n+1)}\sum_{i=1}^{n} i\,p(k-1,i-1)$$

Setting

$$G_n(x) = \sum_{k=1}^{\infty} p(k,n)x^k$$

we have

$$G_n(x) = \sum_{k=1}^{\infty}\left[\frac{2}{n(n+1)}\sum_{i=1}^{n} i\,p(k-1,i-1)\right]x^k$$

It now appears that we are in trouble, for not only do we have two indices in the recurrence, but, in addition, the index upon which we based the definition of the generating function will not help with the factors i or $[1/n(n+1)]$. In order to get such assistance, we define

$$F = \sum_{n=0}^{\infty} G_n(x)y^n$$

$$A = \frac{\partial}{\partial y}(yF) = \sum_{n=1}^{\infty} nG_{n-1}y^{n-1}$$

$$\frac{A}{1-y} = \sum_{n=1}^{\infty}\sum_{i=1}^{n} iG_{i-1}y^{n-1}$$

$$\int\frac{A\,dy}{1-y} = \sum_{n=1}^{\infty}\sum_{i=1}^{n} iG_{i-1}\frac{y^n}{n}$$

$$\iint\frac{A\,d^2y}{1-y} = \sum_{n=1}^{\infty}\sum_{i=1}^{n} iG_{i-1}\frac{y^{n+1}}{n(n+1)}$$

$$= \sum_{n=1}^{\infty}\sum_{i=1}^{n} i\sum_{k=0}^{\infty}\frac{p(k,i-1)x^ky^{n+1}}{n(n+1)}$$

Notice how we have approached, step by step, the form we wish on the right hand side. Each step utilizes a standard technique for generating functions, and the result is very close to what we need; all that is required is a factor of two and an adjustment of the x and y indices:

$$\frac{2x}{y}\iint\frac{A\,d^2y}{1-y} = \sum_{n=1}^{\infty}\left[\sum_{k=1}^{\infty}\left[\sum_{i=1}^{n}\frac{2ip(k-1,i-1)}{n(n+1)}\right]x^k\right]y^n$$
$$= F$$

Differentiating:

$$A=\frac{1-y}{2x}\frac{\partial}{\partial y}A$$

$$\frac{1}{A}\frac{\partial A}{\partial y}=\frac{2x}{(1-y)}$$

$$A=\frac{c}{(1-y)^{2x}}$$

$$=c\sum_{n=0}^{\infty}\binom{-2x}{n}(-y)^{n}$$

where $\binom{-2x}{n}$ is a "generalized" binomial coefficient[11] defined by

$$\binom{-2x}{n}=(-1)^{n}\frac{2x(2x+1)(2x+2)\ldots(2x+n-1)}{n!}.$$

Therefore, using the fact that $G_1(x)=x$ to find $c=1$,

$$A=\frac{\partial}{\partial y}yF=1+xy+\frac{(2x)(2x+1)y}{2}y^2+\cdots$$

$$yF=\sum_{n=0}^{\infty}\binom{-2x}{n}\frac{(-y)^{n+1}}{n+1}$$

$$F=\sum_{n=0}^{\infty}\frac{1}{n+1}\binom{2x+n-1}{n}y^{n}$$

and

$$G_n(x)=\frac{1}{n+1}\binom{2x+n-1}{n}.$$

Why go to all this trouble? Observe that

$$G_n{}'(x)=\frac{d}{dx}G_n(x)=\sum_{n=0}^{\infty}kp(k,n)x^{k-1}.$$

This means that $G_n{}'(1)=\text{mean }(k)$. Similarly

$$G_n{}''(x)=\sum_{n=0}^{\infty}k(k-1)p(k,n)x^{k-2}$$

and

$$G_n{}''(1)=\text{variance}\quad(k)-\text{mean}\quad(k)+\text{mean}\quad{}^2(k)$$

or

$$\text{variance}\quad(k)=G_n{}''(1)+G'(1)-[G'(1)]^2.$$

This provides the motivation; when the generating function is a generating function for a probability distribution, the moment information can be obtained easily from it as shown. A particularly fortuitous occurrence is the occasional emergence of a tractable recurrence involving the derivative of the generating function itself which enables one to arrive at an expected value and a variance without ever obtaining a closed form for the generating function (cf. Reference 12, *Vol. I*, p. 99), but this did not happen here.

Pursuing our analysis, we find that

$$G_n{}'(x)=\frac{2}{(n+1)!}\sum_{i=0}^{n-1}\frac{2x(2x+1)\ldots(2x+n-1)}{2x+i}$$

or

$$G_n{}'(1)=2\sum_{i=1}^{n}\frac{1}{(i+1)}\approx 2[\ln(n+1)-1]$$

Thus, strangely enough, as n grows larger this "random probing" insertion requires only about (2 ln 2) or less than 1.4 times as many compares on the average as does binary search. We shall leave it as a simple mathematical exercise to complete the computation and verify that

$$\text{variance}\quad(k)=2\sum_{i=1}^{n}\frac{1}{i+1}-4\sum_{i=1}^{n}\left(\frac{1}{i+1}\right)^2.$$

So, in this instance, average performance is fairly typical, since the variance is nearly equal to the mean.

By comparison, the "worst case" performance for this algorithm requires comparison of the new item with each of the $(n-1)$ others. Thus, here again, there is a long tail to the distribution. This is not surprising in light of the close relationship between this procedure and Quicksort, the precise nature of which we shall leave to the reader to discover for himself.

As a final remark, we observe that the factor of (2 ln 2) which appears in both the analysis just completed and that of Quicksort is characteristic of binary tree-based random algorithms. As we have seen, the technique of basing the course of action of an algorithm upon operations made on a randomly selected argument is rather more powerful than might at first be apparent [cf. Reference 6].

SUMMARY

In this brief space, it has of course been impossible to do justice to even the nascent field of combinatory algorithm analysis. We have, however, attempted to indicate at least in a qualitative way both the types of information usually sought (and why) and the general character of the accompanying analysis. By far the most authoritative single current source in the areas of performance is the Knuth series,[12] but having the techniques and ideas in hand, there is a broad range of more classical literature to which one may turn for assistance [e.g., References 4, 9, 11, 14, 17].

This is an area of very rapid growth. As it matures, one would expect to see a considerable broadening of analytic technique, but more important will be (a) results which increase our understanding of the principles of algorithm design analogous to, but deeper than, our earlier remark about randomized binary tree algorithms, and (b) results which broaden the scope of possible analysis—such as a good definition(s) of a "random" graph.

REFERENCES

1 F E ALLEN
Program optimization
In Ann Rev Auto Programming *V* Pergamon Press New York 1969

2 K E BATCHER
Sorting networks and their applications
AFIPS Proc SJCC 1968 pp 307-314

3 E W DIJKSTRA
A note on two problems in connection with graphs
Num Mathematik I pp 269-71 1959

4 W FELLER
An introduction to probability theory and its applications
Vol 1 New York John Wiley and Sons 1950

5 R W FLOYD
Assigning meanings to programs
Proc Symp Appl Math 19 J Schwartz Ed Providence Rhode Island Amer Math Soc 1967 pp 19-32

6 W D FRAZER A C McKELLAR
Samplesort: A random sampling approach to minimal storage tree sorting
JACM 17 3 pp 496-507 July 1970

7 T N HIBBARD
Some combinatorial properties of certain trees with applications to searching and sorting
JACM 9 pp 13-28 January 1962

8 C A R HOARE
Quicksort
Computer J 5 1962 pp 10-15

9 C JORDAN
Calculus of finite differences
2nd Ed New York Chelsea Pub 1947

10 J C KING
A program verifier
PhD Dissertation Carnegie-Mellon University 1969

11 K KNOPP
Theory and application of infinite series
Blackie and Sons Ltd London 1928

12 D E KNUTH
The art of computer programming
I, II, III 1968 1969 1972 Addison Wesley Reading Massachusetts

13 ______
Mathematical analysis of algorithms
Proc Cong IFIP-71 To appear

14 L M MILNE-THOMPSON
The calculus of finite differences
London McMillan and Co 1933

15 R C PRIM
Shortest connecting networks and some generalizations
Bell Syst Tech Journal 36 pp 1389-1401 1957

16 E M REINGOLD
Establishing lower bounds on algorithms, a survey
This Proceedings

17 J RIORDAN
An introduction to combinatorial analysis
New York John Wiley and Sons 1958

18 H R STRONG
Translating recursion equations into flowcharts
J Comp and Syst Sci 5 3 pp 254-285 1971

19 P F WINDLEY
Trees, forests, and rearranging
Computer J 3 2 pp 84-88 1960

On the complexity of proving functions*

by ANDY N. C. KANG

University of California
Berkeley, California

INTRODUCTION

Let f be a recursive function. We shall be interested in the following question: given x and y, how difficult is it to decide whether $f(x)=y$ or $f(x)\neq y$? Since the problem of deciding $f(x)=y$ or $f(x)\neq y$ is the same problem as that of computing the characteristic function C_f of the graph of f, we can study the above question by looking into the complexity of computing C_f.

We say that algorithm j serves to prove (or disprove) $f(x)=y$ if $\phi_j^{(2)}\in R_2$ and $\phi_j^{(2)}$ computes the characteristic function, C_f, of the graph of f and $\phi_j^{(2)}(x,y)=1$ $(\phi_j^{(2)}(x,y)=0)$.

We shall show that the complexity of computing functions and the complexity of proving them are approximately equal modulo some recursive function h. Let g and f be recursive functions. We say that "f is difficult to prove almost everywhere (infinitely often) modulo g" if every algorithm which computes C_f takes more than $g(x,y)$ steps to output a 1 for almost all x (infinitely many x) and all y. We say that "f is difficult to disprove almost everywhere (infinitely often) modulo g" if every algorithm which computes C_f takes more than $g(x,y)$ steps to output a 0 for almost all x (infinitely many x) and at least one y.

Based on these definitions, we prove the following results: (1) A function is difficult to prove infinitely often if and only if it is difficult to disprove infinitely often. (2) There exists a function which is difficult to prove almost everywhere, but, surprisingly, it is not difficult to disprove almost everywhere. (3) There exists a function which is difficult to disprove almost everywhere, but it is not difficult to prove almost everywhere.

Before proceeding with our study, we give some preliminaries. Let R_n be the set of recursive functions of n variables. Let $\phi_0^{(2)}, \phi_1^{(2)}, \ldots$ be an acceptable Godel numbering of all the partial recursive functions of two variables [Ref. 4]. A partial recursive function $\Phi_i^{(2)}$, the "step counting function," is associated with each $\phi_i^{(2)}$. The set of partial recursive functions $\{\Phi_i^{(2)}\}_{i\geq 0}$ is arbitrary save for two axioms:
(1) $\phi_i^{(2)}(x,y)$ converges $\rightleftarrows$ $\Phi_i^{(2)}(x,y)$ converges, and
(2) the function

$$M(i,x,y,z)=\begin{cases}1 & \text{if } \Phi_i^{(2)}(x,y)=z\\ 0 & \text{otherwise}\end{cases}$$

is recursive.

Intuitively $\Phi_i^{(2)}(x,y)$ represents the amount of time (or space) used by program i when it finally halts after receiving inputs x and y.

THE COMPLEXITY OF COMPUTING VERSUS PROVING FUNCTIONS

Definition 1:

Let $f\in R_1$. $\phi_k^{(2)}$ is a *characteristic function* for the graph of f if

$$\phi_k^{(2)}(x,y)=\begin{cases}1 & \text{if } f(x)=y\\ 0 & \text{if not}\end{cases}$$

We write $\phi_k^{(2)}=C_f$, where C_f is the *characteristic function* for the graph of f.

Definition 2:

$\Phi_k^{(2)}(x,y)$ is the complexity of algorithm k of *disproving* $f(x)=y$ if $\phi_k^{(2)}=C_f$ and $\phi_k^{(2)}(x,y)=0$.

Definition 3:

$\Phi_k^{(2)}(x,y)$ is the complexity of algorithm k of *proving* $f(x)=y$ if $\phi_k^{(2)}=C_f$ and $\phi_k^{(2)}(x,y)=1$.

The following lemma asserts that the complexity of computing $f(x)$ and the complexity of proving $f(x)=y$ are approximately equal modulo some recursive function h.

* Research sponsored by the National Science Foundation, Grant GJ-708.

Lemma 1:

Let Φ be any complexity measure. There exist three functions $h\in R_3$, $\gamma\in R_1$, $\sigma\in R_1$ such that for any given $f\in R_1$:

(a) If $\phi_i=f$, then $\phi_{\gamma(i)}^{(2)}=C_f$ and

$$\overset{\infty}{\forall}x\forall y[f(x)=y\rightarrow\Phi_{\gamma(i)}^{(2)}(x,y)\leq h(x,y,\Phi_i(x))]$$

(b) If $\phi_k^{(2)}=C_f$, then $\phi_{\sigma(k)}=f$ and

$$\overset{\infty}{\forall}x\forall y[f(x)=y\rightarrow\Phi_{\sigma(k)}(x)\leq h(x,y,\Phi_k^{(2)}(x,y))]$$

Proof:

(a) Any algorithm to compute f can obviously be used to obtain an algorithm for proving f. The complexity of the algorithm for proving f is then bounded by the complexity of the algorithm for computing f. Formally: let γ be a recursive function such that:

$$\phi_{\gamma(i)}^{(2)}(x,y)=\begin{cases}1 & \text{if } \phi_i(x) \text{ converges and } \phi_i(x)=y\\ 0 & \text{if } \phi_i(x) \text{ converges and } \phi_i(x)\neq y\\ \text{diverge} & \text{otherwise}\end{cases}$$

If $\phi_i=f$, then $\phi_{\gamma(i)}^{(2)}$ is recursive since ϕ_i always converges. Then $\phi_{\gamma(i)}^{(2)}(x,y)=1\rightleftarrows\phi_i(x)=y$, i.e., $\phi_{\gamma(i)}^{(2)}=C_f$.

Let

$$p_1(i,x,y,z)=\begin{cases}\Phi_{\gamma(i)}^{(2)}(x,y) & \text{if } \Phi_i(x)=z \text{ and } \phi_i(x)=y\\ 0 & \text{otherwise}\end{cases}$$

p_1 is recursive, since $\lambda xz[\Phi_i(x)=z]$ is a recursive predicate, and $\phi_i(x)$ convergent implies $\Phi_{\gamma(i)}^{(2)}(x,y)$ converges.

Define $h_1(x,y,z)=\max\{p_1(i,x,y,z)\mid i\leq x\}$.

Then we have that for all i:

$$\overset{\infty}{\forall}x\forall y[\phi_{\gamma(i)}^{(2)}(x,y)=1\rightarrow\phi_i(x)\text{ converges and }\phi_i(x)=y\rightarrow h_1(x,y,\Phi_i(x))\geq\Phi_{\gamma(i)}^{(2)}(x,y)].$$

If $\phi_i=f$. Then

$$\overset{\infty}{\forall}x\forall y[f(x)=y\rightarrow\Phi_{\gamma(i)}^{(2)}(x,y)\leq h_1(x,y,\Phi_i(x))].$$

(b) Given an oracle $\phi_k^{(2)}$ for the graph of f, a program can compute f by asking questions about $\phi_k^{(2)}$. Because the graph of f is single-valued, an affirmative answer from $\phi_k^{(2)}$ gives the value of f. Formally: let σ be a recursive function such that:

$\phi_{\sigma(k)}(x)=$"With input x, compute $\phi_k^{(2)}(x,0)$, $\phi_k^{(2)}(x,1)$, ... by dovetailing, until a y appears such that $\phi_k^{(2)}(x,y)=1$; let the output be y."

If $\phi_k^{(2)}$ computes C_f. Then $\phi_{\sigma(k)}$ is recursive and

$$\forall x[\phi_{\sigma(k)}(x)=y\rightleftarrows\phi_k^{(2)}(x,y)=1\rightleftarrows f(x)=y]$$

whence $\phi_{\sigma(k)}=f$.

Let

$$p_2(k,x,y,z)=\begin{cases}\Phi_{\sigma(k)}(x) & \text{if } \Phi_k^{(2)}(x,y)=z \text{ and } \phi_k^{(2)}(x,y)=1\\ 0 & \text{otherwise}\end{cases}$$

p_2 is recursive, since $\lambda xyz[\Phi_k^{(2)}(x,y)=z]$ is a recursive predicate, and $\phi_k^{(2)}(x,y)=1$ implies $\Phi_{\sigma(k)}(x)$ converges. Define $h_2(x,y,z)=\max\{p_2(k,x,y,z)\mid k\leq x\}$

Then we have that for all k:

$$\overset{\infty}{\forall}x\forall y[\phi_k^{(2)}(x,y)=1\rightarrow\Phi_{\sigma(k)}(x)\leq h_2(x,y,\Phi_k^{(2)}(x,y))].$$

If $\phi_k^{(2)}=C_f$. Then

$$\overset{\infty}{\forall}x\forall y[f(x)=y\rightarrow\Phi_{\sigma(k)}(x)\leq h_2(x,y,\Phi_k^{(2)}(x,y))].$$

h in the lemma is defined by

$$h(x,y,z)=\max\{h_1(x,y,z),h_2(x,y,z)\}$$

for all x, y, and z.

Notice that h is independent of the choice of f.

Let (ϕ,Φ) be the class of multitape Turing machines with step counting measure. Then the function h of lemma 1 is roughly given by $h(x,y,z)\approx z^3$. This can be done by a straightforward construction of a Turing machine.

We will now show that a recursive f exists for which there is no lower bound on the complexity for proving $f(x)=y$. This result basically follows from the speed-up theorem [Ref. 2] and lemma 1.

Theorem 1:

Let Φ be any complexity measure. For all $r\in R_1$ there is a $0-1$ valued recursive function $f\in R_1$ such that:

$$\forall k(\phi_k^{(2)}=C_f)\exists j(\phi_j^{(2)}=C_f)\overset{\infty}{\forall}x\forall y[f(x)=y\rightarrow\Phi_k^{(2)}(x,y)>r(\Phi_j^{(2)}(x,y))]$$

Proof:

Without loss generality, we assume r to be monotone increasing. Let $h\in R_3$ be any sufficiently large recursive function monotone increasing in the third variable (h need only be large enough to satisfy lemma 1). Define

$$r'(x,z)=\max\{h(x,y,r(h(x,y,z)))\mid y=0,1\} \tag{1}$$

By the speed-up theorem, there exists a $0-1$ valued function $f\in R_1$ such that:

$$\forall i(\phi_i=f)\exists l(\phi_l=f)\overset{\infty}{\forall}x[\Phi_i(x)>r'(x,\Phi_l(x))] \tag{2}$$

Given $\phi_k^{(2)}=C_f$, it follows from lemma 1 that there exists an i such that:

$$\phi_i = f$$

and

$$\overset{\infty}{\forall} x \forall y [f(x) = y \rightarrow \Phi_i(x) \leq h(x, y, \Phi_k^{(2)}(x, y))] \quad (3)$$

Then by Eq. (2) there is an l such that $\phi_l = f$ and

$$\overset{\infty}{\forall} x [\Phi_i(x) > r'(x, \Phi_l(x))] \quad (4)$$

Applying lemma 1 again, we get $j = \gamma(l)$ such that:

$$\phi_j^{(2)} = C_f$$

and

$$\overset{\infty}{\forall} x \forall y [f(x) = y \rightarrow \Phi_j^{(2)}(x, y) \leq h(x, y, \Phi_l(x))] \quad (5)$$

Therefore we have:

$$\begin{aligned} \overset{\infty}{\forall} x \forall y [f(x) = y \rightarrow h(x, y, \Phi_k^{(2)}(x, y)) & \\ \geq \Phi_i(x) \quad & \text{(by Eq. (3))} \\ > r'(x, \Phi_l(x)) \quad & \text{(by Eq. (4))} \\ \geq h(x, y, r(h(x, y, \Phi_l(x)))) \quad & \text{(by Eq. (1) and the fact that } y = f(x) \in \{0, 1\}) \\ \geq h(x, y, r(\Phi_j^{(2)}(x, y)))] \quad & \text{(by Eq. (5) and the fact that } r \text{ is monotone increasing)} \end{aligned}$$

This implies that

$$\overset{\infty}{\forall} x \forall y [f(x) = y \rightarrow \Phi_k^{(2)}(x, y) > r(\Phi_j^{(2)}(x, y))],$$

since h is monotone increasing in the third variable.

Although the faster program exists, we cannot effectively get such a program. This fact follows from the theorem in Ref. (3).

THE COMPLEXITY OF PROVING VERSUS DISPROVING FUNCTIONS

Next we are going to investigate the complexity for proving and disproving functions. First we make a definition.

Definition 4:

Let $g \in R_2$, $f \in R_1$. We say f is "difficult to prove almost everywhere (infinitely often) modulo g" if every $\phi_i^{(2)}$ computing C_f has the property that for almost all x (infinitely many x) and all y $f(x) = y$ implies $\Phi_i^{(2)}(x, y) > g(x, y)$.

Let $f \in R_1$. For each x there is only one y with the property that $y = f(x)$. Hence it is ambiguous to say f is difficult or easy to disprove at x, since the complexity of disproving f at x also depends on argument y. For example, Rabin's $0-1$ valued recursive function f [Ref. 1] is difficult to compute, hence by lemma 1, f is difficult to prove. As to the complexity of disproving f: if $y \notin \{0, 1\}$, then we know immediately that $f(x) \neq y$. However, if $y \in \{0, 1\}$, then $f(x) \neq y$ is not easy to verify; it is easy to show that this must be the case, i.e., if $y \in \{0, 1\}$, then to prove $f(x) = y$ or to disprove $f(x) = y$ is of the same difficulty.

Since functions, like Rabin's $0-1$ valued function, are generally treated as difficult, we have the following definition for the complexity of disproving functions.

Definition 5:

Let $g \in R_2$, we say that f is "easy to disprove almost everywhere (infinitely often) modulo g" if some $\phi_i^{(2)} = C_f$ has the property that for almost all x (infinitely many x) and all y,[*] $f(x) \neq y \rightarrow \Phi_i^{(2)}(x, y) \leq g(x, y)$.

The complement of this definition is:

Definition 6:

Let $g \in R_2$, we say that f is "difficult to disprove almost everywhere (infinitely often) modulo g" if every $\phi_i^{(2)} = C_f$ has the property that for almost all x (infinitely many x) and some y, $f(x) \neq y$ and $\Phi_i^{(2)}(x, y) > g(x, y)$.

The following theorem asserts that if function f is difficult to prove infinitely often, then it is also difficult to disprove infinitely often.

Theorem 2:

Let Φ be any complexity measure. There is an $h \in R_3$ such that for every step counting function $g \in R_2$ and for all $f \in R_1$, if f is difficult to prove infinitely often modulo $\lambda xy[h(x, y, g(x, y))]$, then f is difficult to disprove infinitely often modulo g.

Proof:

Assume to the contrary that there is a program for C_f, which disproves f easily almost everywhere modulo g, i.e.,

$$\exists j_0 (\phi_{j_0}^{(2)} = C_f) \overset{\infty}{\forall} x \forall y [f(x) \neq y \rightarrow \Phi j_0^{(2)}(x, y) \leq g(x, y)]. \quad (1)$$

Then we may use this program to construct a program which easily proves f almost everywhere. This relies on g being a step counting function. With inputs x and y, our program simply computes $g(x, y)$, then starts computing the given program $\phi_{j_0}^{(2)}(x, y)$. When

[*] If this were "almost all y," then every f would be easy to disprove almost everywhere.

x is sufficiently large, if the computation of $\phi_{j_0}{}^{(2)}(x, y)$ takes more than $g(x, y)$ steps, then we know $f(x)=y$ immediately. Otherwise $\phi_{j_0}{}^{(2)}(x, y)$ takes less than $g(x, y)$ steps to converge. In that case we can prove or disprove $f(x)=y$ according to the value of $\phi_{j_0}{}^{(2)}(x, y)$. Now we are going to give a formal proof of the above outline.

Let α be a recursive function such that:

$$\phi_{\alpha(i,j,l)}{}^{(2)}(x, y)=\begin{cases} \phi_{j(x,y)}{}^{(2)} & \text{if } x<l \text{ or} \\ & (x\geq l \text{ and } \phi_i{}^{(2)}(x, y) \text{ converges} \\ & \text{and } \Phi_j{}^{(2)}(x, y)\leq\Phi_i{}^{(2)}(x, y)) \\ 1 & \text{if } x\geq l \text{ and } \phi_i{}^{(2)}(x, y) \\ & \text{converges and} \\ & \Phi_j{}^{(2)}(x, y)>\Phi_i{}^{(2)}(x, y) \\ \text{diverge} & \text{otherwise} \end{cases}$$

Lemma:

If $g=\Phi_{i_0}{}^{(2)}$ and if $j=j_0$ is given by Eq. (1), then for some l_0, $\phi_{\alpha(i_0,j_0,l_0)}{}^{(2)}=C_f$.

Proof:

Since $\phi_{i_0}{}^{(2)}$ and $\phi_{j_0}{}^{(2)}$ are total, $\phi_{\alpha(i_0,j_0,l)}{}^{(2)}$ is total. By Eq. (1) there is a number l_0 such that for all x and all y, if $x\geq l_0$ and $f(x)\neq y$, then $\Phi_{j_0}{}^{(2)}(x, y)\leq g(x, y)$. Therefore $\phi_{\alpha(i_0,j_0,l_0)}{}^{(2)}$ computes C_f. End of lemma.

Now we shall construct the h of the theorem. Define: $p(i, j, l, x, y, z)$

$$=\begin{cases} 0 & \text{if } x<l \\ \Phi_{\alpha(i,j,l)}{}^{(2)}(x, y) & \text{if } x\geq l \text{ and } \Phi_i{}^{(2)}(x, y)=z \\ 0 & \text{otherwise} \end{cases}$$

p is recursive, for if $x\geq l$ and $\Phi_i{}^{(2)}(x, y)$ converges, then $\Phi_{\alpha(i,j,l)}{}^{(2)}(x, y)$ converges.

Let

$$h(x, y, z)= \max\{p(i, j, l, x, y, z) \mid \max\{i, j, l\}\leq x\}.$$

Thus we have that for all i, j, l:

$$\overset{\infty}{\forall} x \forall y[h(x, y, \Phi_i{}^{(2)}(x, y))\geq\Phi_{\alpha(i,j,l)}{}^{(2)}(x, y)].$$

Hence, particularly for $i=i_0$, $j=j_0$, $l=l_0$:

$$\overset{\infty}{\forall} x \forall y[h(x, y, g(x, y))\geq\Phi_{\alpha(i_0,j_0,l_0)}{}^{(2)}(x, y)],$$

since $\Phi_{i_0}{}^{(2)}(x, y)=g(x, y)$.

This contradicts the hypothesis that f is difficult to prove infinitely often modulo $\lambda xy[h(x, y, g(x, y))]$, since by the lemma: $\phi_{\alpha(i_0,j_0,l_0)}{}^{(2)}$ is a program that computes C_f.

Conversely, we can show that if f is difficult to disprove infinitely often, then it is also difficult to prove infinitely often.

Theorem 3:

Let Φ be any complexity measure. There is an $h\in R_3$ such that for every step counting function $g\in R_2$ and for all $f\in R_1$, if f is difficult to disprove infinitely often modulo $\lambda xy[h(x, y, g(x, y))]$, then f is difficult to prove infinitely often modulo g.

Proof:

The proof is similar to the proof of theorem 2. Notice that if there is a program $\phi_j{}^{(2)}$ for C_f which proves f easily almost everywhere modulo g, then by definition this means that for almost all x there is a y such that $f(x)=y$ and $\Phi_j{}^{(2)}(x, y)\leq g(x, y)$. Since the graph of f is single-valued, if x is large and $\Phi_j{}^{(2)}(x, y)>g(x, y)$, then $f(x)$ cannot possibly be equal to y.

Theorems 2 and 3 show that a function is difficult to prove infinitely often if and only if it is difficult to disprove infinitely often.

Naturally, we may ask the question: if $f(x)=y$ is difficult to prove for almost all x, then does it follow that for each such x there is a y such that $f(x)=y$ is also difficult to disprove? This is false by the following theorem.

Theorem 4:

Let Φ be any complexity measure. There is a function $b\in R_2$ such that for all function $g\in R_2$, there exists a function $f\in R_1$ with two properties:

(a) f is difficult to prove almost everywhere modulo g.

(b) f is easy to disprove infinitely often modulo b.

Proof:

The theorem follows from the fact that even if a function f has the property that there is a bound $b(x, y)$ on the number of steps to disprove $f(x)=y$ for infinitely many x and all y, it may still be difficult to locate these x for which $f(x)=y$ is easy to disprove.

Let $g\in R_2$ be given. Let $h\in R_3$ be any sufficiently large function monotone increasing in the third variable (h need only be large enough to satisfy lemma 1). Let $\lambda x[a(x)]$ be a $0-1$ valued recursive function such that:

$$\forall i(\phi_i=a)\overset{\infty}{\forall} x[\Phi_i(x)>h(x, 0, g(x, 0))] \qquad (1)$$

The existence of $\lambda x[a(x)]$ follows from Rabin's theorem [Ref. 1].

Let ϕ_{i_0} be a fixed program for $\lambda x[a(x)]$. We define f as follows: $f(x)$: "With input x, first compute $f(y)$ for all $y<x$. (In the process of doing this, some finite set of ϕ_i will be cancelled.) Second compute $\phi_{i_0}(x)$.

Case 1: $\phi_{i_0}(x)=0$, let the output be 0.

Case 2: $\phi_{i_0}(x)=1$, look for $j_0=\mu j[j\leq x$ and j is not cancelled and $\Phi_j(x)\leq 1+\max\{h(x, y, g(x, y))\mid y\in\{\Phi_{i_0}(x)+1, \Phi_{i_0}(x)+2\}\}$.

If no such j_0 is found, let the output be $\Phi_{i_0}(x)+1$. Otherwise, let the output be $\Phi_{i_0}(x)+2$ if $\phi_{j_0}(x)=\Phi_{i_0}(x)+1$. Let the output be $\Phi_{i_0}(x)+1$ if $\phi_{j_0}(x)\neq\Phi_{i_0}(x)+1$.

Then cancel ϕ_{j_0} from the standard list."

Let us see why f works:

(b) We know that for any sufficiently difficult $0-1$ function a, $a(x)=0$ for infinitely many x. From the construction of f, $a(x)=0$ implies $f(x)=0$. We shall exhibit a program which has the property that it disproves $f(x)=y$ quickly whenever $a(x)=0$ and $y\neq 0$. In fact, we will define a recursive function b, which does not depend on f such that b is an upper bound for the number of steps to disprove $f(x)=y$ whenever $a(x)=0$ and $y\neq 0$.

First we construct $\phi_{\sigma(i,k)}{}^{(2)}$. We will show that $\phi_{\sigma(i,k)}{}^{(2)}$ is the program that disproves f quickly when k is an index for f and i is i_0, where ϕ_{i_0} is the program used in the construction of f.

Let σ be a recursive function such that:

$$\phi_{\sigma(i,k)}{}^{(2)}(x, y)=\begin{cases} 1\dot{-}\phi_k(x) & \text{if } y=0 \text{ and } \phi_k(x) \text{ converges}\\ 0 & \text{if } y\neq 0 \text{ and}\\ & \quad(\Phi_i(x)>y \text{ or } \phi_i(x)=0\\ & \text{or } v(\phi_k(x) \text{ converges and}\\ & \quad \phi_k(x)\neq y))\\ 1 & \text{if } y\neq 0 \text{ and } \Phi_i(x)\leq y \text{ and}\\ & \quad \phi_i(x)\neq 0 \text{ and } \phi_k(x) \text{ converges and } \phi_k(x)=y\\ \text{diverge} & \text{otherwise}\end{cases}$$

Lemma 4.1:

If k is an index for f and if ϕ_{i_0} is the program used in the construction of f, then $\phi_{\sigma(i_0,k)}{}^{(2)}=C_f$.

Proof:

Observe that ϕ_k is total implies that $\phi_{\sigma(i_0,k)}{}^{(2)}$ is total.

Case 1: $y=0$.

$$\phi_{\sigma(i_0,k)}(x, y)=\begin{cases} 1 & \text{if } \phi_k(x)=0\\ 0 & \text{if } \phi_k(x)\neq 0\end{cases}$$

$$\phi_{\sigma(i_0,k)}(x, 0)=C_f(x, 0) \text{ for all } x.$$

Case 2: $y\neq 0$.

Subcase 1: $\Phi_{i_0}(x)>y$. In this subcase,

$$\phi_{\sigma(i_0,k)}{}^{(2)}(x, y)=0.$$

Now look at the construction of f. $f(x)\in\{0, \Phi_{i_0}(x)+1, \Phi_{i_0}(x)+2\}$. Therefore $f(x)$ will either be 0 or will be greater than y. Therefore $f(x)\neq y$.

Subcase 2: $\Phi_{i_0}(x)\leq y$ and $\phi_{i_0}(x)=0$. Again,

$$\phi_{\sigma(i_0,k)}{}^{(2)}(x, y)=0.$$

Look at the program for f. $\phi_{i_0}(x)=0$ implies $f(x)=0$, therefore: $f(x)\neq y$.

Subcase 3: $\Phi_{i_0}(x)\leq y$ and $\phi_{i_0}(x)\neq 0$ and $\phi_k(x)=y$ implies $\phi_{\sigma(i_0,k)}(x, y)=1 \therefore$ $\phi_{\sigma(i_0,k)}{}^{(2)}$ gives the correct answer. End of lemma 4.1.

Define b as follows:

$$b(x, y)=\begin{cases}\max\{\Phi_{\sigma(i,k)}{}^{(2)}(x, y)\mid i\in A \text{ and } k\leq x\}\\ \qquad \text{if } y\neq 0 \text{ and } A\neq\emptyset, \text{ where}\\ \qquad A=\{i\mid i\leq x \text{ and } [\Phi_i(x)>y \text{ or } \phi_i\\ 0 \qquad (x)=0]\} \text{ otherwise}\end{cases}$$

b is recursive since one can effectively decide whether or not $A=\emptyset$, and in case $y\neq 0$ and $A\neq\emptyset$, then by definition of σ, $\phi_{\sigma(i,k)}{}^{(2)}(x, y)=0$, so $\Phi_{\sigma(i,k)}{}^{(2)}(x, y)$ converges for all x.

Lemma 4.2:

$$\forall i\forall k\overset{\infty}{\forall} x\forall y[\phi_i(x)=0$$

and

$$y\neq 0\rightarrow\Phi_{\sigma(i,k)}{}^{(2)}(x, y)\leq b(x, y)] \quad (2)$$

Proof:

Let $x\geq\max\{i, k\}$. If $\phi_i(x)=0$, then i will be in A. If i is in A and $y\neq 0$, then $b(x, y)\geq\Phi_{\sigma(i,k)}{}^{(2)}(x, y)$. End of lemma 4.2.

Let $i=i_0$, and let k be an index for f. By lemma 4.1, $\phi_{\sigma(i_0,k)}{}^{(2)}=C_f$. By lemma 4.2,

$$\overset{\infty}{\forall} x\forall y[\phi_{i_0}(x)=0$$

and

$$y\neq 0\rightarrow\Phi_{\sigma(i_0,k)}{}^{(2)}(x, y)\leq b(x, y)].$$

By the definition of f, we know that $\phi_{i_0}(x)=0\rightleftarrows f(x)=0$. Because ϕ_{i_0} is a sufficiently complex $0-1$ valued function, we know $\phi_{i_0}(x)=0$ for infinitely many x. Hence, we have:

$$\overset{\infty}{\exists} x\forall y[f(x)\neq y\rightarrow\Phi_{\sigma(i,k)}{}^{(2)}(x, y)\leq b(x, y)].$$

This finishes the proof of part (b).

The proof of part (a) is easier: Let ϕ_k be any program which computes f.

Lemma 4.3:

$$\overset{\infty}{\forall} x[\Phi_k(x)>h(x, f(x), g(x, f(x)))] \quad (3)$$

Proof:

Case 1: $f(x)=0$

Assume to the contrary that

$$\overset{\infty}{\exists}x[\Phi_k(x)\le h(x,0,g(x,0))]$$

By the definition of f, we know $f(x)=0\rightleftarrows\phi_{i_0}(x)=0$. By assumption, program ϕ_k computes f in less than $h(x,0,g(x,0))$ steps infinitely often when $f(x)=0$. Hence, it computes $\phi_{i_0}(x)$ in less than $h(x,0,g(x,0))$ steps infinitely often when $\phi_{i_0}(x)=0$. This is a contradiction, since

$$\forall i(\phi_i=\phi_{i_0})\overset{\infty}{\forall}x[\Phi_i(x)>h(x,0,g(x,0))],$$

by Eq. (1).

Case 2: $f(x)\neq 0$.

Assume to the contrary that

$$\overset{\infty}{\exists}x[\Phi_k(x)\le h(x,f(x),g(x,f(x)))].$$

Notice that $f(x)\neq 0$ implies $f(x)\in\{\Phi_{i_0}(x)+1, \Phi_{i_0}(x)+2\}$. Look at the definition of f: there are infinitely many chances for k to be cancelled. Since at most k programs with index less than k may be cancelled, ϕ_k will eventually be cancelled and the value of f will be made different from ϕ_k. This contradicts $\phi_k=f$. End of lemma 4.3.

Let $\phi_i^{(2)}=C_f$. By lemma 1, there is a k such that $\phi_k=f$ and

$$\overset{\infty}{\forall}x\forall y[f(x)=y\rightarrow\Phi_k(x)\le h(x,y,\Phi_i^{(2)}(x,y))] \qquad (4)$$

Combining Eq. (3) and Eq. (4), we have:

$$\overset{\infty}{\forall}x\forall y[f(x)=y\rightarrow h(x,y,\Phi_i^{(2)}(x,y)) \ge\Phi_k(x)>h(x,y,g(x,y))].$$

If follows that:

$$\overset{\infty}{\forall}x\forall y[f(x)=y\rightarrow\Phi_i^{(2)}(x,y)>g(x,y)],$$

since h is monotone increasing in the third variable.

This finishes the proof of part (a).

Let (ϕ, Φ) be the class of multitape Turing machines with step counting measure. Then the function b of theorem 4 is given by $b(x,y)=C\cdot y$, where C is a constant. To see this, we simply present a program to prove $f(x)\neq y$ in $C\cdot y$ steps when $\phi_{i_0}(x)=0$ (note that $\phi_{i_0}(x)=0$ occurs for infinitely many x), informally: given inputs x and y, if $y>0$, run y steps of $\phi_{i_0}(x)$. If $\phi_{i_0}(x)$ does not converge in y steps, we know immediately $f(x)\neq y$ (since $f(x)\in\{0, \Phi_{i_0}(x)+1, \Phi_{i_0}(x)+2)\}$). Otherwise $\phi_{i_0}(x)$ converges in y steps, and in case $\phi_{i_0}(x)=0$, then we conclude that $f(x)\neq y$ (since $\phi_{i_0}(x)=0\leftrightarrow f(x)=0$).

Parallel to the question asked before theorem 4, we may ask the question: if f is difficult to disprove almost everywhere, is it also difficult to prove almost everywhere? The answer is again negative by a similar construction of f as in theorem 4.

Theorem 5:

Let Φ be any complexity measure. There is a function $b\in R_2$ such that for all function $g\in R_3$, there exists a function $f\in R_1$ with two properties:

(a) f if difficult to disprove almost everywhere modulo g.
(b) f is easy to prove infinitely often modulo b.

Proof:

The proof is similar to that of theorem 4. First we define a function $h_1\in R_4$, the reason we define h_1 will be clear later.

Let δ be a recursive function such that

$$\phi_{\delta(i,j)}{}^{(2)}(x,y):$$

"With input x and y. Compute $\phi_i(x)$. If and when $\phi_i(x)$ converges. Check following 3 cases.

Case 1: $\phi_i(x)=0$; give output $1\dot{-}|y-\Phi_i(x)|$
Case 2: $\phi_i(x)=1$; compute $\phi_j{}^{(2)}(x,0)$ and $\phi_j{}^{(2)}(x,1)$ simultaneously until one of them converges.

Subcase 1: $\phi_j{}^{(2)}(x,0)=0$ or $\phi_j{}^{(2)}(x,1)=1$; give output $1\dot{-}|y-1|$

Subcase 2: $\phi_j{}^{(2)}(x,0)=1$ or $\phi_j{}^{(2)}(x,1)=0$; give output $1\dot{-}y$

Subcase 3: $\phi_j{}^{(2)}(x,0)$ converges and $\phi_j{}^{(2)}(x,0)\notin\{0,1\}$ or $\phi_j{}^{(2)}(x,1)$ converges and $\phi_j{}^{(2)}(x,1)\notin\{0,1\}$; give output 0

Case 3: $\phi_i(x)$ converges and $\phi_i(x)\notin\{0,1\}$; give output 0."

Define:

$$h_1(x,y,w,z)=\begin{cases}\max\{\Phi_{\delta(i,j)}{}^{(2)}(x,y)\mid\langle i,j\rangle\in A\}\\ \quad\text{if } A\neq\phi\text{, where}\\ A=\{\langle i,j\rangle\mid i\le x\text{ and } j\le x\\ \quad\text{and }\Phi_i(x)=w\text{ and}\\ \quad(\Phi_j{}^{(2)}(x,0)\le z\text{ or }\Phi_j{}^{(2)}(x,1)\le z)\}\\ 0\quad\text{otherwise}\end{cases}$$

h_1 is recursive, since we can effectively decide whether $\langle i,j\rangle\in A$ or not. And in case $\langle i,j\rangle\in A$, $\Phi_{\delta(i,j)}{}^{(2)}(x,y)$ converges.

Lemma 5.1:

$$\forall i\forall j\overset{\infty}{\forall}x\forall y[\phi_i(x)\text{ converges and }(\phi_j{}^{(2)}(x,0)\text{ converges or }\phi_j{}^{(2)}(x,1)\text{ converges})\rightarrow\Phi_{\delta(i,j)}{}^{(2)}(x,y)\le h_1(x,y,\Phi_i(x),\min\{\Phi_j{}^{(2)}(x,0),\Phi_j{}^{(2)}(x,1)\})]$$

Proof:

Let $x \geq \max\{i, j\}$. Since either

$$\Phi_j^{(2)}(x, 0) \leq \min\{\Phi_j^{(2)}(x, 0), \Phi_j^{(2)}(x, 1)\}$$

or

$$\Phi_j^{(2)}(x, 1) \leq \min\{\Phi_j^{(2)}(x, 0), \Phi_j^{(2)}(x, 1)\},$$

by the definition of A we know that $\langle i, j\rangle \in A$. Therefore, for all $x \geq \max\{i, j\}$ and all y

$$[\Phi_{\delta(i,j)}^{(2)}(x, y) \leq h_1(x, y, \Phi_i(x), \min\{\Phi_j^{(2)}(x, 0), \Phi_j^{(2)}(x, 1)\})].$$

End of lemma 5.1.

Now we are going to define f. Let h_1 be any sufficiently large function monotone increasing in the fourth variable (h_1 need only be large enough to satisfy lemma 5.1 above). Let $g \in R_2$ be given. Let $h \in R_3$ be any sufficiently large function monotone increasing in the third variable (h need only be large enough to satisfy lemma 1). Let $\lambda x[a(x)]$ be a $0-1$ valued recursive function such that:

$$\forall i(\phi_i = a) \overset{\infty}{\forall} x[\Phi_i(x) > 2 + \max\{g(x, 0), g(x, 1)\}]. \quad (1)$$

The existence of $\lambda x[a(x)]$ follows from Rabin's theorem [Ref. 1].

Let ϕ_{i_0} be a fixed program for $\lambda x[a(x)]$. We define f as follows: $f(x)$: "With input x, first compute $f(y)$ for all $y < x$. (In the process of doing this, some finite set of ϕ_i will be cancelled.) Second compute $\phi_{i_0}(x)$.

Case 1: $\phi_{i_0}(x) = 0$, let the output be $\Phi_{i_0}(x)$.

Case 2: $\phi_{i_0}(x) = 1$, look for $j_0 = \mu j[j \leq x$ and j is not cancelled and

$$\Phi_j(x) \leq \max_{y \in \{0,1\}} \{h(x, y, \max_{w \in \{0,1\}} \{h_1(x, w, \Phi_{i_0}(x), \max\{g(x, 0), g(x, 1)\})\})\}]$$

If no such j_0 is found, let the output be 0. Otherwise, let the output be $1 \dot{-} \phi_{j_0}(x)$. Then cancel ϕ_{j_0} from the standard list."

Let us see why f works:

(a) We will show that for all sufficiently large x, there exists some y such that $f(x) \neq y$ and $\Phi_i^{(2)}(x, y) > g(x, y)$ for every $\phi_i^{(2)}$ computing C_f.

Lemma 5.2:

For all sufficiently large x such that $\phi_{i_0}(x) = 0$, $f(x) = y$ is difficult to disprove modulo g for some y. i.e.,

$$\forall j(\phi_j^{(2)} = C_f) \overset{\infty}{\forall} x[\phi_{i_0}(x) = 0 \rightarrow \exists y[f(x) \neq y$$

and

$$\Phi_j^{(2)}(x, y) > g(x, y)]].$$

Proof:

Assume to the contrary that: $\phi_{j_0}^{(2)} = C_f$ and

$$\overset{\infty}{\exists} x[\phi_{i_0}(x) = 0$$

and

$$\forall y[f(x) \neq y \rightarrow \Phi_{j_0}^{(2)}(x, y) \leq g(x, y)] \quad (2)$$

From the construction of f, we know that $\phi_{i_0}(x) = 0 \leftrightarrow f(x) > 1$. Therefore, if we can decide $f(x) > 1$ in less than $\max\{g(x, 0), g(x, 1)\}$ steps for infinitely many x, then we can compute $\phi_{i_0}(x)$ in less than $\max\{g(x, 0), g(x, 1)\}$ steps for infinitely many x. This will be a contradiction to Eq. (1). Since $\phi_{j_0}^{(2)}$ computes C_f, a way to decide whether $f(x) > 1$ or not is to compute $\phi_{j_0}^{(2)}(x, 0)$ and $\phi_{j_0}^{(2)}(x, 1)$ simultaneously. For those x satisfying Eq. (2), the computation of $\phi_{j_0}^{(2)}(x, 0)$ and $\phi_{j_0}^{(2)}(x, 1)$ will converge in less than $g(x, 0)$ and $g(x, 1)$ steps respectively, since by the definition of f for those x with the property that $\phi_{i_0}(x) = 0$. We know that $f(x) \neq 0$ and $f(x) \neq 1$. Therefore, we can decide $f(x) > 1$ in less than $\max\{g(x, 0), g(x, 1)\}$ steps for infinitely many x. End of lemma 5.2.

Lemma 5.3:

For all sufficiently large x such that $\phi_{i_0}(x) = 1$, $f(x) = y$ is difficult to disprove modulo $g(x, y)$ for some y. i.e.,

$$\forall_j(\phi_j^{(2)} = C_f) \overset{\infty}{\forall} x[\phi_{i_0}(x) = 1 \rightarrow \exists_y[f(x) \neq y$$

and

$$\Phi_j(x, y) > g(x, y)]]$$

Proof:

Claim:

For all sufficiently large x such that $\phi_{i_0}(x) = 1$, $f(x) = y$ is difficult to prove modulo

$$\lambda x[\max_{w \in \{0,1\}} \{h_1(x, w, \Phi_{i_0}(x), \max\{g(x, 0), g(x, 1)\})\}]$$

for all y. i.e.,

$$\forall_j(\phi_j^{(2)} = C_f) \overset{\infty}{\forall} x[\phi_{i_0}(x) = 1$$
$$\rightarrow \forall y[f(x) = y \rightarrow \Phi_j^{(2)}(x, y)$$
$$> \max_{w \in \{0,1\}} \{h_1(x, w, \Phi_{i_0}(x),$$
$$\max\{g(x, 0), g(x, 1)\})\}]]$$

Proof:

First we show that: for every ϕ_k computing f

$$\overset{\infty}{\forall} x[\phi_{i_0}(x) = 1 \rightarrow [\Phi_k(x)$$
$$> \max_{y \in \{0,1\}} \{h(x, y, \max_{w \in \{0,1\}} \{h_1(x, w, \Phi_{i_0}(x),$$
$$\max\{g(x, 0), g(x, 1)\})\})\}]] \quad (3)$$

Since otherwise there is a ϕ_k such that $\phi_k = f$ and

$$\overset{\infty}{\exists} x[\phi_{i_0}(x) = 1$$

and

$$\Phi_k(x) \leq \max_{y\in\{0,1\}} \{h(x, y, \max_{w\in\{0,1\}} \{h_1(x, w, \Phi_{i_0}(x), \max\{g(x, 0), g(x, 1)\})\})\}].$$

Look at the definition of f, then there are infinitely many chances for k to be cancelled and ϕ_k will eventually be cancelled and the value of f will be made different from ϕ_k. This contradicts that ϕ_k computes f.

Second, let $\phi_j^{(2)} = C_f$. By lemma 1 there is a k such that $\phi_k = f$ and

$$\overset{\infty}{\forall} x \forall y[f(x) = y \rightarrow \Phi_k(x) \leq h(x, y, \Phi_j^{(2)}(x, y))] \quad (4)$$

Therefore, we have:

$$\overset{\infty}{\forall} x[\phi_{i_0}(x) = 1 \rightarrow \forall y[f(x) = y$$
$$\rightarrow h(x, y, \Phi_j^{(2)}(x, y)) \geq \Phi_k(x)$$
$$\text{(by Eq. (4))}$$
$$\geq h(x, y, \max_{w\in\{0,1\}} \{h_1(x, w, \Phi_{i_0}(x), \max\{g(x, 0), g(x, 1)\})\})]]$$
$$\text{(by Eq. (3) and } \phi_{i_0}(x) = 1 \rightarrow y = f(x) \in \{0, 1\})$$

It follows that:

$$\overset{\infty}{\forall} x[\phi_{i_0}(x) = 1 \rightarrow \forall y[f(x) = y \rightarrow \Phi_j^{(2)}(x, y) \geq \max_{w\in\{0,1\}} \{h_1(x, w, \Phi_{i_0}(x), \max\{g(x, 0), g(x, 1)\})\}]];$$

since h is monotone in the third variable. End of claim.

Next we assume to the contrary to lemma 5.3 that there is a $\phi_{j_0}{}^{(2)}$ computes C_f has the property that:

$$\overset{\infty}{\exists} x[\phi_{i_0}(x) = 1$$

and

$$\forall y[f(x) \neq y \rightarrow \Phi_{j_0}(x, y) \leq g(x, y)]] \quad (5)$$

We will use ϕ_{j_0} to construct a program which proves $f(x) = y$ in less than

$$\max_{w\in\{0,1\}} \{h_1(x, w, \Phi_{i_0}(x), \max\{g(x, 0), g(x, 1)\})\}$$

steps for infinitely many x such that $\phi_{i_0}(x) = 1$. Then this is a contradiction to the claim above.

We claim $\phi_{\delta(i_0, j_0)}{}^{(2)}$ will do: from the definition of δ, it is easy to see that $\phi_{\delta(i_0,j_0)}{}^{(2)}$ computes C_f, since ϕ_{j_0} computes C_f and ϕ_{i_0} is used in defining f.

Since $\phi_{j_0}{}^{(2)}$ is total and $\phi_{i_0}(x) = 1$ implies that $f(x) \in \{0, 1\}$. From lemma 5.1 and Eq. (5) above we have the result that for infinitely many x such that $\phi_{i_0}(x) = 1$ and for $y \in \{0, 1\}$,

$$\Phi_{\delta(i_0,j_0)}{}^{(2)}(x, y) \leq h_1(x, y, \Phi_{i_0}(x),$$
$$\min\{\Phi_{j_0}(x, 0), \Phi_{j_0}(x, 1)\}) \leq \max_{w\in\{0,1\}} h_1(x, w, \Phi_{i_0}(x),$$
$$\min\{\Phi_{j_0}(x, 0), \Phi_{j_0}(x, 1)\}) \leq \max_{w\in\{0,1\}} h_1(x, w, \Phi_{i_0}(x),$$
$$\max\{g(x, 0), g(x, 1)\})$$

(since h_1 is monotone increasing in the fourth variable and either $\Phi_{j_0}{}^{(2)}(x, 0) \leq g(x, 0)$ or $\Phi_{j_0}{}^{(2)}(x, 1) \leq g(x, 1)$ by Eq. (5)).

This is a contradiction to the claim above. End of lemma 5.3.

Combining lemma 5.2 and lemma 5.3, we have the proof of part (a).

(b) We will construct $\phi_{\sigma(i,k)}{}^{(2)}$. When we choose $i = i_0$ and k to be an index for f, then $\phi_{\sigma(i_0,k)}{}^{(2)}$ computes C_f. In addition, we will show that there is a recursive b, which does not depend on f and g such that b is an upper bound for the number of steps to compute $\phi_{\sigma(i_0,k)}{}^{(2)}(x, y)$ whenever $\phi_{i_0}(x) = 0$ and $y = \Phi_{i_0}(x)$. Look at the definition of f, $f(x) = \Phi_{i_0}(x)$ when $\phi_{i_0}(x) = 0$. Therefore b is an upper bound for the number of steps to prove $f(x) = y$ by algorithm $\sigma(i_0, k)$ when $\phi_{i_0}(x) = 0$. Since ϕ_{i_0} is a sufficiently complex $0-1$ valued function, we know $\phi_{i_0}(x) = 0$ for infinitely many x.

Let σ be a recursive function such that:

$$\phi_{\sigma(i,k)}{}^{(2)}(x, y)$$
$$= \begin{cases} 1 \dot{-} \mid y - \phi_k(x) \mid & \text{if } y \leq 1 \text{ and } \phi_k(x) \text{ converges} \\ 1 & \text{if } y > 1 \text{ and } \Phi_i(x) = y \\ & \quad \text{and } \phi_i(x) = 0 \\ 0 & \text{if } y > 1 \\ & \quad \text{and } (\Phi_i(x) > y \text{ or } (\Phi_i(x) = y \\ & \quad \text{and } \phi_i(x) \neq 0) \text{ or } \Phi_i(x) < y) \\ \text{diverge} & \text{otherwise} \end{cases}$$

Lemma 5.4:

If k is an index for f and if ϕ_{i_0} is the program used in the construction of f, then $\phi\sigma_{(i_0,k)}{}^{(2)} = C_f$.

Proof:

Observe that ϕ_k total implies that $\phi_{\sigma(i_0,k)}{}^{(2)}$ is total. It is obvious to see $\phi_{\sigma(i_0,k)}{}^{(2)}(x, y) = C_f(x, y)$ when $y \leq 1$. By definition of f, $f(x) \in \{0, 1, \Phi_{i_0}(x)\}$ and $f(x) = \Phi_{i_0}(x)$ only when $\phi_{i_0}(x) = 0$. End of lemma 5.4.

Define b as follows:

$$b(x,y)=\begin{cases}\max\{\Phi_{\sigma(i,k)}(x,y) \mid k\leq x \\ \quad \text{and } i\in A\} \text{ if } A\neq\emptyset, \text{ where} \\ A=\{i \mid i\leq x \text{ and } y>1 \\ \quad \text{and } \Phi_i(x)=y \text{ and } \phi_i(x)=0\} \\ 0 \quad \text{otherwise}\end{cases}$$

b is recursive since one can effectively decide whether or not $A=\emptyset$, and in case $A\neq\emptyset$, then by definition of σ, $\phi_{\sigma(i,k)}{}^{(2)}(x,y)=1$. So $\Phi_{\sigma(i,k)}{}^{(2)}(x,y)$ converges.

Lemma 5.5:

$$\forall i\forall k\overset{\infty}{\forall} x\forall y[y>1 \text{ and } \phi_i(x)=0 \text{ and } \Phi_i(x)=y\rightarrow\Phi_{\sigma(i,k)}{}^{(2)}(x,y)\leq b(x,y)].$$

Proof:

Let $x\geq\max\{i,k\}$. If $y>1$ and $\phi_i(x)=0$ and $\Phi_i(x)=y$, then i will be in A then $b(x,y)\geq\Phi_{\sigma(i,k)}{}^{(2)}(x,y)$. End of lemma 5.5.

Let $i=i_0$, and let k be an index for f. By lemma 5.4, $\phi_{\sigma(i_0,k)}{}^{(2)}=C_f$. Because ϕ_{i_0} is a sufficiently complex $0-1$ valued function, we know $\phi_{i_0}(x)=0$ for infinitely many x. By definition of f, $\phi_{i_0}(x)=0$ implies $f(x)=\Phi_{i_0}(x)$ and $\Phi_{i_0}(x)>1$ by Eq. (1). Hence by lemma 5.5, we have:

$$\overset{\infty}{\exists} x\exists y[f(x)=y \text{ and } \Phi_{\sigma(i_0,k)}{}^{(2)}(x,y)\leq b(x,y)]$$

i.e., we show that f is easy to prove infinitely often modulo b. This ends the proof of part (b).

ACKNOWLEDGMENTS

The author is deeply indebted to Professor M. Blum for suggesting the topic and close supervision throughout the whole study, especially in the proof of theorem 4.

REFERENCES

1 M RABIN
Degree of difficulty of computing a function and a partial ordering on recursive sets
Technical Report No 2 Hebrew University April 1960

2 M BLUM
A machine independent theory of computational complexity
JACM 14 322-336 1967

3 *On effective procedure for speeding up algorithms*
ACM Symposium on Theory of Computing Marina del Ray May 1969

4 H ROGERS
Theory of recursive functions and effective computability
McGraw-Hill New York 1967

On the structure of Blum measure*

by TSUN S. CHOW

University of California
Berkeley, California

PRELIMINARIES

We assume that the reader is familiar with the basic paper of Blum,[1] and Borodin's paper on the existence of complexity gaps.[2]

$\mathcal{N}$ is the set of nonnegative integers. $\mathcal{R}_n$ is the set of (total) recursive functions of n variables. $\mathcal{P}_n$ is the set of partial recursive functions of n variables.

The abbreviation "a.e." is used for "almost everywhere." If $P(x)$ is a statement containing the variable x then $\lambda x[P(x)]$ is a predicate of one variable on $\mathcal{N}$. The λ-notation is also used for functions. The statement "$\lambda x[P(x)]$ (a.e.)" means that $P(x)$ is true for all but finitely many $x \in \mathcal{N}$. Similarly, "$\lambda xy[P(x, y)]$ (a.e.)" means that $P(x, y)$ is true for all but finitely many pairs of x and $y \in \mathcal{N}$. The phrase "for sufficiently large function $f \in \mathcal{R}_n \ldots$" means "there is a $b \in \mathcal{R}_n$ such that for all $f, f \geq b$ (a.e.)$\Rightarrow \ldots$."

The function ϕ_i is then ith partial recursive function in a standard enumeration of $\mathcal{P}_i$. A Blum *measure* $\Phi = \{\Phi_0, \Phi_1, \ldots\}$ is a sequence of functions in $\mathcal{P}_1$ satisfying two axioms:

1. domain $(\phi_i) =$ domain (Φ_i) for all $i \in \mathcal{N}$, and
2. $\lambda i, x, y[\Phi_i(x) = y]$ is a recursive predicate.

As an aid in exposition, we say f is less (greater) than g, *modulo* r to mean that $rf < g$ a.e. ($f > rg$ a.e.).

Choose a recursive 1-1 map of the integers onto the set of all 2-tuples of integers. Let $\langle x, y \rangle$ denote the integer which maps onto (x, y).

"DENSENESS" OF BLUM MEASURE

Borodin[2] has shown that given any recursive function r, there exists a recursive t such that no step-counting function takes values between $t(x)$ and $rt(x)$ except on a finite number of integers. The gap, here, appears between two recursive functions, t and rt.

At this point, a natural question to ask is the following: for every recursive function r, does there exist a pair of recursive step-counting functions Φ_i and Φ_j such that $\Phi_j(x) > r(x, \Phi_i(x))$ for almost all x and there are no step-counting functions between them? Corollary will show that, in fact, for all sufficiently large recursive function r, there can be no such gaps between any two recursive step-counting functions, or even any two *partial* recursive step-counting functions with same but infinite domain.

The following proposition shows that for any partial recursive function ϕ_i there exists another program u, which computes ϕ_i but takes slightly more steps than that required to compute ϕ_i.

Proposition

There exist functions $\sigma \in \mathcal{R}_1$ and $h \in \mathcal{R}_2$ such that for every i, we have

(a) $\phi_{\sigma(i)} = \phi_i$ and
(b) $\Phi_i(x) < \Phi_{\sigma(i)}(x) \leq h(x, \Phi_i(x))$
a.e. in the domain of ϕ_i.

Proof:

Let σ be defined as follows:

$$\phi_{\sigma(i)}(x) = \begin{cases} \text{diverge} & \text{if} \quad \phi_i(x) \quad \text{diverges} \\ & \text{or} \quad \Phi_{\sigma(i)}(x) \leq \Phi_i(x) \\ \phi_i(x) & \text{otherwise} \end{cases}$$

Clearly σ is recursive and $\phi_{\sigma(i)} = \phi_i$. Also, for all x in the domain of ϕ_i, we have $\Phi_i(x) < \Phi_{\sigma(i)}(x)$.

Let p and h be defined as follows:

$$p(i, x, z) = \begin{cases} \Phi_{\sigma(i)}(x) & \text{if} \quad \Phi_i(x) = z \\ 0 & \text{if otherwise} \end{cases}$$

$$h(x, z) = \max_{i \leq x} \{p(i, x, z)\}$$

* Research sponsored by the National Science Foundation, Grant GJ-708

Clearly p is recursive and so is h. Further we have for almost all x in the domain of ϕ_i

$$\Phi_{\sigma(i)}(x) \leq h(x, \Phi_i(x))$$

Corollary:

Let $r \in \mathcal{R}_2$ be sufficiently large. For every i, j, if

(a) domain (ϕ_i) = domain $(\phi_j) = W$ and
(b) $\Phi_j(x) > r(x, \Phi_i(x))$ a.e. in W

then there is a u such that

(c) $\phi_u = \phi_i$ and
(d) $\Phi_i(x) < \Phi_u(x) < \Phi_j(x)$ a.e. in W.

Proof:

Let σ and h be as in Proposition (2.1). Assume r be such that $r > h$ a.e.

For every i, j satisfying (a) and (b) above, we have that for almost all x in the domain of ϕ_i

by Proposition (2.1) $\Phi_i(x) < \Phi_{\sigma(i)} \leq h(x, \Phi_i(x))$
by choice of r $< r(x, \Phi_i(x))$
by choice of j $< \Phi_j(x)$

Further $\phi_{\sigma(i)} = \phi_i$

Hence $\sigma(i)$ is the u desired by the corollary.

"TOP" AND "BOTTOM" OF GAPS

There is another slightly weaker way to talk about gaps. With $h \in \mathcal{R}_2$ fixed, pick a recursive function r much greater than h. Then we say that "Φ_i is the top (bottom) of a gap of size r," if whenever Φ_k is less (greater) than Φ_i modulo h a.e. then Φ_k is much less (greater) than Φ_i, in the sense that Φ_k is less (greater) than Φ_i modulo r.

Now we can ask the following question: does there exist an $h \in \mathcal{R}_2$ such that for every recursive r there is some Φ_i which is the top (bottom) of a gap of size r? Theorem (3.1) says that there exists a Φ_i which serves for the top of a gap, while for sufficiently large r, Theorem (3.2) says that no Φ_i can serve as the bottom of a gap of size r.

Theorem (*Top of the gap theorem*)

There exists an $h \in \mathcal{R}_2$ such that for every r, $g \in \mathcal{R}_1$, there is a u uniform in r and g with the following properties:

(a) Φ_u is recursive and
(b) $\forall x[\Phi_u(x) > g(x)]$ and

(c) $\forall k \overset{\infty}{\forall} x[h(x, \Phi_k(x)) < \Phi_u(x)$
$\rightarrow r(\langle x, \Phi_k(x) \rangle) < \Phi_u(x)]$

Proof:

Instead of working directly with r, $r = \phi_i$ and g, $g = \phi_j$, we choose to work with $\Phi_{\alpha(i)}$ (in place of ϕ_i) and $\Phi_{\alpha(j)}$ (in place of ϕ_j). Besides majorizing the recursive functions, these $\{\Phi_{\alpha(n)}\}_{n \geq 0}$ have other useful properties.

Lemma 1:

Let Φ be a Blum measure. There is a recursive function α such that for every i, x, y

(a) $\Phi_{\alpha(i)}(x) \geq \phi_i(x)$
(b) $\Phi_{\alpha(i)}(\langle x, y \rangle) > y$
(c) if $\Phi_{\alpha(i)}(\langle x, y \rangle)$ and $\Phi_{\alpha(i)}(\langle x, y+1 \rangle)$ converge, then $\Phi_{\alpha(i)}(\langle x, y \rangle) < \Phi_{\alpha(i)}(\langle x, y+1 \rangle)$
(d) $\Phi_{\alpha(i)}$ recursive iff ϕ_i recursive.
(e) if $\Phi_{\alpha(i)}(\langle x, y \rangle)$ diverges for some y, then for all $z \geq y$ $\Phi_{\alpha(i)}(\langle x, z \rangle)$ diverges

Lemma 2:

With α as in Lemma 1, there is $\delta \in \mathcal{R}_2$ such that

(a) for every i, j, x $\phi_{\delta(i,j)}(x) \geq \Phi_{\alpha(j)}(x)$
(b) $\lambda ijxz[\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x) \rangle) = z]$ is recursive
(c) if ϕ_i, ϕ_j are recursive, then $\phi_{\delta(i,j)}$ is recursive and has the following property:

$$\forall k \forall x \geq k[\neg[\phi_{\delta(i,j)}(x) \leq \Phi_k(x) \leq \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x) \rangle)]]$$

Lemma 2(b) is needed to make the construction of h in Lemma 4 become recursive.

Lemma 3:

With α as in Lemma 1 and δ as in Lemma 2, there exists $\sigma \in \mathcal{R}_2$ such that for every i, j, x

(a) $\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x) \rangle)$ converges if and only if $\Phi_{\sigma(i,j)}(x)$ converges and
(b) $\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x) \rangle) \leq \Phi_{\sigma(i,j)}(x)$

Lemma 4:

With α, δ, and σ as in Lemma 1, 2, 3 respectively, there exists an $h \in \mathcal{R}_2$, monotonically increasing on the second variable such that

$$\forall i, j \forall x \geq \max\{i, j\}[h(x, \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x) \rangle)) \geq \Phi_{\sigma(i,j)}(x)]$$

Now we are ready to prove the theorem. The function h is given by Lemma 4. Let recursive function $r = \phi_i$ and $g = \phi_j$ be given. We choose $u = \sigma(i, j)$. Since ϕ_i, ϕ_j recursive imply, by Lemma 1(d) and Lemma 2(c), that $\Phi_{\alpha(i)}$ and $\phi_{\delta(i,j)}$ are recursive, we conclude, by Lemma 3(a), that $\Phi_{\sigma(i,j)}$ recursive. This proves (a) of the theorem.

From Lemma 3(b), we have

$$\Phi_{\sigma(i,j)}(x) \geq \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x) \rangle) \quad \text{for every } x.$$

Recalling Lemma 1(b), Lemma 2(a), and Lemma 1(a), this implies

$$\Phi_{\sigma(i,j)}(x) > \phi_j(x) = g(x) \quad \text{for every } x.$$

This proves (b) of the theorem.

It remains to show that $\sigma(i,j)$ also satisfies (c) of the theorem.

Let Φ_k be such that for some $x > \max\{i, j, k\}$

$$h(x, \Phi_k(x)) < \Phi_{\sigma(i,j)}(x)$$

By Lemma 2(c), $\phi_{\delta(i,j)}$ is recursive. Since by Lemma 4, for $x \geq \max\{i, j\}$:

$$\Phi_{\sigma(i,j)}(x) \leq h(x, \Phi_{\alpha(i)}(\langle x, \Phi_{\delta(i,j)}(x)\rangle))$$

we have

$$h(x, \Phi_k(x)) < h(x, \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle))$$

Since h is monotonically increasing on the second variable, this implies

$$\Phi_k(x) < \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle)$$

by Lemma 2(c)

$$\Phi_k(x) < \phi_{\delta(i,j)}(x)$$

by Lemma 1(c)

$$\Phi_{\alpha(i)}(\langle x, \Phi_k(x)\rangle) < \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle) \; (*)$$

Since by Lemma 1(a),

$$\phi_i(\langle x, \Phi_k(x)\rangle) < \Phi_{\alpha(i)}(\langle x, \Phi_k(x)\rangle).$$

and by Lemma 3(b),

$$\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle) < \Phi_{\sigma(i,j)}(x)$$

the inequality (*) implies

$$\phi_i(\langle x, \Phi_k(x)\rangle) < \Phi_{\sigma(i,j)}(x)$$

That is, we have shown that for every k and almost all x,

$$h(x, \Phi_k(x)) < \Phi_{\sigma(i,j)}(x) \rightarrow \phi_i(\langle x, \Phi_k(x)\rangle) < \Phi_{\sigma(i,j)}(x)$$

Proof of Lemma 1:

It is easy to construct a recursive function α satisfying Lemma 1 and so the proof is omitted here.

Proof of Lemma 2:

Given $\Phi_{\alpha(i)}$ and $\Phi_{\alpha(j)}$, then for each x we wish to set $\phi_{\delta(i,j)}(x) = y$ for the least y such that y is greater than $\Phi_{\alpha(j)}(x)$ and there is no $\Phi_k(x)$ between y and $\Phi_{\alpha(i)}(\langle x, y\rangle)$ for all $k \leq x$. If we know in advance that both $\Phi_{\alpha(i)}$ and $\Phi_{\alpha(j)}$ are recursive, then this is exactly like the proof of Borodin's Gap Theorem. But in general, $\Phi_{\alpha(i)}$ and $\Phi_{\alpha(j)}$ may not converge at some inputs and therefore we may not be able to define $\phi_{\delta(i,j)}$ at some x. Hence, in order to make the predicate $\lambda ijxz[\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle) = z]$ recursive, we have to take extra trouble in defining $\phi_{\delta(i,j)}(x)$.

We first introduce a recursive predicate Q and then define $\phi_{\delta(i,j)}(x)$ in terms of Q, such that if $Q(i, j, x, z)$ is true for some z, then $\phi_{\delta(i,j)}(x)$ converges.

$$Q(i, j, x, z) \equiv \exists y[(1)\ \Phi_{\alpha(j)}(x) < y$$

and

$$(2)\ \Phi_{\alpha(i)}(\langle x, y\rangle) = z$$

and

$$(3)\ \forall k \leq x[\neg(y \leq \Phi_k(x) \leq z = \Phi_{\alpha(i)}(\langle x, y\rangle))]$$

Since Φ is a Blum measure and by Lemma 1(c) and 1(e) for every i, $\Phi_{\alpha(i)}$ is strictly increasing on its domain on second variable, we can check effectively whether there exists a y satisfying conditions (1) and (2) above. If there is such a y, then we can check condition (3). Thus we see that Q is recursive.

Now define

$$\phi_{\delta(i,j)}(x) = \begin{cases} \text{diverge} & \text{if } \forall z[\neg Q(i, j, x, z)] \\ y & \text{otherwise,} \\ & \text{where } y \text{ is the unique integer} \\ & \quad \text{such that} \\ & \Phi_{\alpha(i)}(\langle x, y\rangle) = \mu z[Q(i, j, x, z)] \end{cases}$$

δ is recursive, because the predicate Q is recursive. By condition (1) in the definition of Q, we have that for every i, j, and x, $\phi_{\delta(i,j)}(x) \geq \Phi_{\alpha(j)}(x)$. This proves (a) of the lemma. Now, for every i, j, x, z:

$$\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle) = z \leftrightarrow Q(i, j, x, z)$$

and

$$\forall w < x[\neg Q(i, j, x, w)]$$

Since the right-hand-side in the above expression is recursive, we see that $\lambda\, i, j, x, z[\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle) = z]$ is also recursive. This proves (b) of the lemma.

To prove (c), we assume ϕ_i and ϕ_j to be total. By Lemma 1(d), we have $\Phi_{\alpha(i)}$ and $\Phi_{\alpha(j)}$ total. By examining the definition Q, we see that for every x, there is always some z such that $Q(i, j, x, z)$ holds. Therefore, $\phi_{\delta(i,j)}(x)$ converges for every x, and so $\phi_{\delta(i,j)}$ is recursive.

Also, from construction of δ, for every x, $\phi_{\delta(i,j)}(x)$ defined implies that

$$\forall k \forall x \geq k[\neg[\phi_{\delta(i,j)}(x) \leq \Phi_k(x) \leq \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle)]]$$

Proof of Lemma 3:

Let α and δ be as in Lemma 1 and Lemma 2 respectively. Let σ be the function defined as follows:

$$\phi_{\sigma(i,j)}(x) = \begin{cases} 0 & \text{if } \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle) \text{ converges} \\ & \quad \text{and } \Phi_{\sigma(i,j)}(x) > \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle) \\ \text{diverge} & \text{otherwise} \end{cases}$$

This definition makes implicit use of the recursion theorem.

Since $\Phi_{\alpha(i)}$ and $\phi_{\delta(i,j)}$ are partial recursive, σ is recursive. From the definition of σ it is clear that $\Phi_{\sigma(i,j)}$ has the same domain as $\lambda x[\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle)]$ and for every x, $\Phi_{\sigma(i,j)}(x)$ exceeds $\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle)$ whenever the latter converges.

Proof of Lemma 4:

Let α, δ, and σ be as in Lemma 1, 2, 3, respectively. Let

$$p(i, j, x, z) = \begin{cases} \Phi_{\sigma(i,j)}(x) & \text{if } \Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle) = z \\ 0 & \text{otherwise.} \end{cases}$$

By Lemma 3, if $\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle)$ converges then $\Phi_{\sigma(i,j)}(x)$ also converges, and by Lemma 2,

$$\lambda i, j, x, z[\Phi_{\alpha(i)}(\langle x, \phi_{\delta(i,j)}(x)\rangle) = z]$$

is recursive. Therefore, p is recursive.

Let

$$h(x, 0) = 1 + \max_{i \leq x, j \leq x} \{p(i, j, x, 0)\}$$

and for $z > 0$

$$h(x, z+1) = 1 + \max_{i \leq x, j \leq x} \{p(i, j, x, z), h(x, z)\}$$

Clearly h has the desired properties.

Theorem (*Bottom of the gap theorem*)

Let $t \in \mathcal{R}_2$ be given. If $r \in \mathcal{R}_2$ is sufficiently large, then for every ϕ_i there exists u such that:

(a) $\phi_u = \phi_i$ and

(b) $t(x, \Phi_i(x)) < \Phi_u(x) < r(x, \Phi_i(x))$ a.e. in the domain of ϕ_i.

Proof:

By a trivial modification in the proof of Proposition, we can show the following lemma:

Lemma:

For every $t \in \mathcal{R}_2$ there exist $\sigma \in \mathcal{R}_1$ and $h \in \mathcal{R}_2$ such that for every i, we have:

(a) $\phi_{\sigma(i)} = \phi_i$ and

(b) $t(x, \Phi_i(x)) < \Phi_{\sigma(i)}(x) < h(x, \Phi_i(x))$ a.e. in the domain of Φ_i.

The theorem follows from the Lemma, if we let $u = \sigma(i)$ and assume r is greater than h a.e.

CONCLUSION

Besides Borodin's Gap Theorem, we have presented two other ways to look at the gap phenomenon in the structure of Blum measure. It is no accident that the proof of Theorem (Top of the Gap Theorem) is closely related to that of Borodin's Gap Theorem while the proof of Theorem (Bottom of the Gap Theorem) is very similar to that of Corollary. In a way, they explain why the gap that exists in Borodin's formulation fails to appear in Corollary. Our main results point out the existence of an interesting asymmetry in the properties of step-counting functions.

ACKNOWLEDGMENTS

The author is deeply indebted to Professor M. Blum for suggesting the topic and close supervision throughout the whole study, especially in the proof of Theorem (Top of the Gap Theorem).

REFERENCES

1 M BLUM
A machine independent theory of computational complexity
JACM 14 pp 322-336 April 1960

2 A BORODIN
Complexity classes of recursive functions and the existence of complexity gaps
ACM Symposium on the Theory of Computing 1969 pp 67-78

Management information systems, public policy and social change

by ABE GOTTLIEB

Office of State Planning and Development
Harrisburg, Pennsylvania

I would like to address a few of my remarks to the character and implications of computer usage in government but before I do so, I might indicate that I'm with the Pennsylvania Office of State Planning and Development and my frame of reference has three outward tracks—as an urban planner, as a participant in data systems and as an official and sometimes unofficial stimulator of social change. These three biases ought to surface in the remarks I am going to make.

State governments as well as national and local ones are basically concerned with two kinds of computer uses. The first, and much less important, puts the computer to work at massaging multi-purpose information that can cover an almost infinite spectrum of urban oriented subject areas. These have come to be known as urban information systems. In theory, they can furnish the planner and decision maker with data and analyses in land use changes, population and housing shifts, economic trends, fiscal information and social statistics for a community or any group of communities.

But computer technology in government plays a much more critical role than what is inherent in the operations and outputs of such urban information systems. It is almost a truism today that no public agency, bureau, commission, board or any other unit of government can operate without some computerized control of its programs, processes and service delivery capabilities. For example, in Pennsylvania each of the 18 executive departments and most of the commissions and boards have their own data systems. Many of these, especially the larger ones, are computerized to a fairly high level of sophistication. The State Departments of Transportation, Labor and Industry, Revenue, Education and Public Welfare all have a tremendous amount of stored information and a commensurate technology to utilize that data.

This kind of operation and use of computers is associated with a relatively new breed called Management Information Systems. Its focus is the day to day, month to month control of inventory, bookkeeping, record keeping, disbursements and receipts in the myriads of government programs that inter-act with each other and with many recipients of public services. My concern therefore, is with the Management Information Systems that are utilized by States and municipalities throughout the country. These systems exist by virtue of controlling an inventory or a process and by sometimes delivering a service to public or private recipients.

INFORMATION SYSTEMS AS MANAGEMENT TOOLS

Now as I get into the substance of my discussion, I want to explore a number of ideas that relate more directly to the relationship of these computerized systems to the bureaucracies in which they flourish, to the formation or paralysis of public policy and to their impact on the possibilities of social change. Since these are the central themes around which our discussion will resolve, I will direct my remarks to the following points:

1. Urban data systems and management information systems are not merely respositories of facts and instructions for manipulating them. They are also collections of technologies as well and if there ever was any truth to the cliche that the "medium is the message," I am almost tempted to make the point that it is really the computer and its applications that will fashion, modify and transform our social structure and not the data that is poured into and out of it.
2. Management information systems at the State, local and national levels are used to serve management. In Pennsylvania State government as elsewhere the data systems are designed to assist in the operation of a Department's or agency's program more efficiently and perhaps more

cheaply than would have been possible under other arrangements.

3. Management rationality and efficiency are the overriding considerations of the "worth" of such systems. Whether to design, install, enlarge, merge with other systems or evaluate, the principal factors are judged to be speed, thoroughness, cost and labor savings. These are managerial-administrative considerations and almost never socially oriented ones. They look inward to the process, not outward to the consequences.
4. The programs or process controls which are the heart of such data systems sometimes result in services to various segments of the State's population. Sometimes they do not and we will speak about both situations shortly.
5. Management information systems that typically extend such services as auto registrations, tax notifications and collections, unemployment and welfare assistance, etc., may or may not be necessary or desirable. They can, however, make little claim to generating, sustaining or otherwise influencing the direction, depth or quality of social change in the State.
6. Any data management system with the capability of mounting substantially greater research and analytic capabilities than was previously possible, can sometimes produce an innovative breakthrough, a new way of seeing and understanding the dynamics of a community. Conceivably, this might lead to important changes and realignments in our social, political and economic structures but I remain skeptical and would like to be convinced.
7. Finally, I feel that if we turned the proposition the other way around and considered the impact of currently changing social values on the information systems and their related technologies, we would get a new dimension to the problem. If the events, moods and changes that surfaced in the latter part of the 1960s mean anything for the coming decades, we can expect major incursions into the organization, content and uses or perhaps even selective abandonment of urban oriented information systems.

THE REACH OF MANAGEMENT INFORMATION SYSTEMS

Now before anyone hands in his union card, I would like to make it clear that while data and management information systems are rarely the catalysts for social change, they can and do penetrate to the lives of a great many people in all segments of our society. This stems directly from the operations of the record keeping and process controls that are at the very heart of the systems. At the State level of government in Pennsylvania such operations probably account for over 90 percent of all computer usage so that it would not be amiss to say that both the administrators of the information systems and the machinery itself is thoroughly committed to keeping the records and controlling the processes. For example, the State Department of Public Education receives a veritable deluge of data from the local school districts relating to almost all aspects of running an educational bureaucracy. The Department of Revenue must maintain a computerized information system for the calculation, billing, mailing, recording and analyzing of tax receipts. Similarly, the information systems of the State Department of Labor and Industry and the Department of Transportation (just to name a few of the larger operations) are deeply committed to the routinized processes of maintaining a complex unemployment insurance system in the first case and keeping auto ownership inventories, issuing licenses and recording accidents in the second.

Basically, therefore, the data systems of the State work towards the delivery of services to a wide range of individual residents as well as to local communities and to other State agencies. There are of course adjacent and parallel operations such as internal payroll, bookkeeping and property inventory and control that generally do not reach out to any segment of the population. One can easily see the counter-part of these operations at the city, county and township levels as well. This is, admittedly, a capsule version of the Pennsylvania Management Information System but I think it is important to appreciate the data areas, the kinds of controlled operations and a few examples of the services delivered before we speak about the roles, if any, that these systems play in social change.

THE LIMITING LIDS ON SOCIAL INNOVATIONS

You may have noticed that I have deliberately divorced the objectives and operation of management information systems from the larger State programs of which they are part. I admit that this is not quite fair and that such a division would be a highly artificial one so I would like to bridge the gap at this point. The State of Pennsylvania has defined eight broad, long range objectives toward which it wants to move and had identified several hundred currently operating programs (many in the social arena) that may or may not be reinforcing these objectives. All the data systems I have been talking about live and operate in some uneasy relationship to these programs and objectives and it is these connections that I would like to discuss.

To begin with, the State information systems that keep records, control processes and deliver some services in Pennsylvania (and there are almost no other kinds) are themselves very substantial organizational entities. The investment in time, professional staff, money hardware and software and support personnel is quite considerable and this whole range of activity has gained structural and administrative solidity in the last few years. In Pennsylvania and in many other States we can speak of an information system bureaucracy that is not without substantial power and influence. Like all bureaucracies, it does not always look kindly upon change inside or outside its own domain; social, managerial or otherwise.

I have also made the point earlier that management information systems are tools to serve management which means more precisely to serve administrators and decision makers in operating programs and process as efficiently, rapidly and cheaply as possible. Now while these are the criteria that make the computer so valuable they are also the criteria that draw the administrator to the narrow focus of the procedural rationality of his program rather than towards an examination and concern with the impacts and consequences of his acts. In other words, the machine together with its operations, controls and data base induce the administrator to look inwards to the managerial objectives of his world when he should be involved with the political, social and economic meaning of what he does. We are very accustomed to visualize information systems as devices to facilitate policies and programs but it seems to me that the feedback (if you will pardon the expression) might be even more significant than the initial impulse that is, the systems themselves (singly and in concert) set the pattern, depth and range of current and future concerns of the State and many municipalities. From this point of view, the tail is very definitely wagging the dog and something is wrong.

I am not intimating that programs made effective by the use of information systems and process controls are unimportant or negative accomplishments in the State. But I am saying that the systems, controls and processes should not be the measuring rod for the social and economic interactions between the State and its residents. Nor should they be (as they almost inevitably become) the determinants that freeze public policy.

CONSTRAINTS ON PUBLIC POLICY

At this point in our discussion, I feel that we are beginning to zero in on a critical area. Almost at the outset, I made the observation that social changes among various groups and segments of our population can usually be traced to a shift in values, attitudes and aspirations that arise in these groups and the conviction that such aspirations can be realized in one's lifetime. However, to these basic stimulants I would like to add a second kind of possibility and that is the nature and scope of public policy.

Under certain circumstances, public policy can translate the values and aspirations of various groups into attainable benefits. Such policies and programs can even set off a chain reaction extending into social, political and economic areas far beyond the point of initial program impact. It is my contention that what happens to existing programs and especially the breadth and vision brought to bear by public administrators in creating new social objectives will go a long way towards making a management information system responsive to social realities. The field is wide open in health, criminal justice, education, consumer protection, public welfare, environmental improvement and others. And it is in these areas and among these kinds of public efforts that I have some qualified reasons to believe that the State may become attuned to the explosive social transformations that are sometimes above and usually just below the surface in the major urban areas of the State.

However, the obstacles to this kind of sequence are formidable. To begin with, public policies, objectives and programs are heavily dependent on what the agency or the administrator is doing now and doing it with all the management tools, personnel and investments at his disposal. Realistically speaking, there is no particular reason to assume that State, city or national management people look forward to gearing themselves, their program control processes, their machines and their data to ventures that cannot be justified beyond running smooth, self-regulating and almost self-perpetuating operations.

For example, it is with considerable fear and trepidation that many administrators look upon program evaluations where the focus is on social, educational and economic consequences, ameliorations and status changes rather than what is expected and dictated by the flow processes of paper, forms and data. This feeling is especially acute when the validity of existing programs and sequenced actions are questioned and new ones might be suggested. The real threat of P.P.B. is not that cost-effectiveness studies are hard to define and difficult to mount (all too true) but that P.P.B. insists that management and its information systems clarify and justify their efforts in terms of its impacts and consequences on all classes and categories of recipients. In the social arenas of criminal justice, poverty, education, housing and even science and technology, this begins to get close to tampering with the cement that holds the structure together and whether we call it P.P.B. or anything else it is clearly a kind of opening wedge in that direction.

Therefore, the way I see it two related things must happen. The first is a willingness for the public policy makers to realign State and local objectives and programs so that definable improvements are explicitly built into them and the second a willingness for the information systems directors to accept new standards, criteria, data and processes to make this work. It could be that the policy makers and the information directors are not always the same people with the same interests and outlooks on all issues of program and process control. Sometimes they wear the same hat and sometimes they don't but it seems reasonable to suppose that they have been mutually reinforcing each other as long as information systems have continued to remain program and management tools. If I were to venture a guess, I would say that the policy and program makers are as much locked into the uniquely narrow perspectives of data, inventory and process control as are the information systems directors themselves.

SOME PROOF OF THE PUDDING-SOCIAL INDICATORS

My *theory* about management being quite allergic to social change might be reinforced by a telling *fact* about management and here I mean the public agencies, State, Federal and local that operate in what we have been calling the social arena. For the brutal fact of the matter is that they have shut ourselves off from the data and intelligence that might tell them just a tiny bit about the social world around us. In "Towards a Social Report" of the Department of Health, Education and Welfare we are made painfully aware of the fact that "the nation has no comprehensive set of statistics reflecting social progress or retrogression. There is no government procedure for stock taking of the social health of the nation."

Somebody has said this much better than I can and I would like to quote from Elizabeth Drew writing in a publication called "The Public Interest":

> *"Those who picture Washington as one mass of files and computers containing more information than they would like will be comforted by the experiences of program-planners in attempting to evaluate on-going programs. Whatever the files and computers do contain, there is precious little in them about how many and whom the programs are reaching, and whether they are doing what they are supposed to do. If the purpose of an adult basic education program is to teach people how to read and write, the Office of Education might reasonably be expected to know how many people thereby actually learned how to read and write but it does not. . . . The Public Health Service might be expected to know whether its various health services are in fact making people healthier but it does not. The study of disease control was to have encompassed more diseases, but so little was known about the effective treatment of alcoholism and heart disease that these components had to be dropped. Those working on the income maintenance study found that the Welfare Administration could not tell them very much about the public assistance case load—who was on welfare, where did they come from, why were they on it; what they needed in order to get off."*

If this is true of the national level and with Federal programs, it is more than doubly so with the States and the localities. Our knowledge vacuum is truly an anomolous situation. It stares us in the face despite the fact (or more correctly because of the fact) that the Federal, State and local governments have amassed and computerized a vast body of statistical information necessary to run their programs and churn their processes. Paradoxically, while we are inundated in a sea of paper, ink and printouts, we cannot measure the human toll of illness, the pollution of the environment, the quality of our education and the nature of the alienation expressed in burning and looting in the ghetto, strife on the campus and crime in the city streets. From everything I have said up till now, management and its information systems want to have no part of these "social" changes except perhaps to wish that they go away.

Geographic information systems in the U.S.—An overview

by ROBERT AMSTERDAM, EDWARD ANDRESEN and HARRY LIPTON

Office of the Mayor—Office of Administration
New York, New York

A Geographic Information System (GIS) can be defined as one which is oriented to supplying information pertaining to the geography or spatial relationships of the information in the system.

When one asks for the total of all school aged children from a welfare file who are in school district 52 or when one wants to affix the health area code to a file of birth or death records, one is involved in relating geographic descriptors (e.g., the school district and health area codes which represent finite, fixed areas of space), to other geographic descriptions (e.g., the address of the welfare child, the home address of the mother . . . etc.).

Information such as the above has only recently become processable via computer. Heretofore, for a file of any significant size, large amounts of manual effort were required in order to enable one to relate one set of geographic descriptors to another. Now, through the use of the principal components of a typical Geographic Information System, these files can be related at computer processing speeds.

The principal components of a typical GIS are:

- Geographic information files (GIF) containing all the geographic descriptors of interest for the geographic area covered
- A program or set of programs that enable the GIF to be accessed (matched) against data files and appropriate geographic descriptor information appended from the "matched" GIF record to the "matched" data file record.
- Graphical display programs and equipment which enable the matched output to be displayed, via computer, in the form of maps or charts in addition to the usual listings.

GEOGRAPHIC INFORMATION FILES

Geographic files can encompass the entire range of local government activities. Two types of files, however, have played a major role in the development of Geographic Information Systems; these are the Address Coding Guide/Geographic Base File and the Parcel/Building File.

Address coding guide/geographic base files

These files were first developed in the early 1960's for regional transportation studies such as the Penn-Jersey Study in the Philadelphia area and the Tri-State Transportation Study in the Metropolitan New York area. The files were used in compiling information on origin and destination of trips by all means of travel within the study area. The results were used in planning new transportation routes, determining patterns of economic and residential growth, and planning for development of new resources. Comparable files were developed by private firms for use in market analysis by commercial organizations. Such files have also been developed by the Post Office to assist in zipcoding. By the mid-1960's, the U. S. Bureau of the Census recognized the value of these files in coordinating a mail-out census and began encouraging their development in all urban areas of the country. Since then their broad range of uses has been increasingly recognized by local government and groups engaged in analysis of small area data. The files have been modified, expanded and updated to meet the demands of local users.

In its basic form, an Address Coding Guide (ACG) relates ranges of addresses to defined geographic areas. Most simply, it could contain the range of building addresses in a traffic zone or postal area. A series of records is prepared for each street name. Each record contains the highest and lowest building numbers on that street which lie within the specific zone. Generally, separate records are prepared for odd and even house number ranges.

As Address Coding Guides have developed, it has been recognized that as more different area designations are incorporated in the file, the more useful it becomes.

Since many operational jurisdictions (e.g., post office routes, police precincts, census enumeration districts) bound whole city blocks, it is usually convenient to create a separate record for each block side. This record contains the street name, highest and lowest building numbers that can be used on that street section, and whatever area codes can be usefully related to that block or block side. Additionally, the record can contain alternate names for that street, the names of intersecting streets, the width of the street, whether it is paved, the type of paving, whether it is privately or publicly owned, direction of traffic, speed limit and other information of general interest

This information can be related to other data files in order to develop a profile of a community, to coordinate operational data or to retrieve specific information contained in the ACG. One significant example of its use is in relating the number of incidents in an area, such as number of births, or crime reports, to the total population in the area in order to calculate the overall birth rate or crime rate for the community. Such data is required for many types of community planning and deployment of facilities.

An important extension of the ACG is the Geographic Base File (GBF). In a GBF, each record can be the same as an ACG record but it also includes x and y coordinates of one or more fixed points on the block related to a grid system which is uniformly referenced for the entire locality. The individual record may contain the coordinates of a corner of the block, the center of the block, the center of the blockside, the intersections of the streets bounding the blockside or some other recognizable point. With this added data, the location of any block or address is determined with reference to any other location in the area. This allows the user to calculate distance, determine the entities in a given perimeter or radius and make other spatial determinations. This capability is important in transportation planning, in setting up routes or district lines and in determining the potential users for a new facility. Grid coordinates are also valuable in the preparation of computer-generated maps. Maps are essential in many local government operations and the ability to produce detailed maps showing the locations of specific properties or addresses is increasingly playing an important part in administrative decisions.

As one example of use of the GBF, complaints of cracks or holes in street paving can be correlated with street segment identification numbers in the GBF. When street repair work is scheduled, crews can be readily informed of all complaints in a compact area so that minimal time is lost in traveling.

ACG's and GBF's are being used in many local areas throughout the United States and Canada. They are also being developed for several major cities in Europe, Israel and South America. Their development requires a considerable investment by the locality to check out and inventory all streets, blocks and significant land features in their jurisdiction and to update the information as changes occur. However these files are critical for the development of many types of municipal information systems and operational activities and the effort is increasingly proving to be justified.

The U. S. Census Bureau is developing a standard format for these files called DIME (for Dual Independent Map Encoding). In a DIME file, each record is related to a street segment. It identifies the street name, address range on each side of the segment, census block on each side of the segment and intersection node coordinates at each end of the segment.[1]

Parcel/building files

The second major file important for local area data collection systems is the Building File. This contains one record for each building in the region with its address, dimensions, floor space, usage, owner's name and whatever additional data the local government finds useful to maintain.

Traditionally, the source for this data is the property assessment department. Assessors are required to maintain a complete inventory of the property in their district in order to insure that all land is taxed and that valuations are reviewed and updated periodically to reflect improvements to the property, improvements to the neighborhood and changes in market values. In most parts of the United States assessment is a county responsibility. Cities lying within a county and metropolitan areas which lie in several counties frequently have administrative problems in gaining access to assessors' files.

There have also been technical obstacles to using this data: Many assessors' files are in detailed, handwritten records which have been accumulated over long periods of time. Data is not standardized and difficult decisions must be made on which data can most usefully be converted to computerized form. Another obstacle has been that assessors usually keep records of land parcels or lots, since this allows the most complete inventory of the land in a county. However this makes it difficult to obtain clear information on individual buildings when there is more than one building on a lot.

One final problem will be mentioned: it is frequently impossible to record the owner of a parcel on a compact, fixed-length field. The property may be owned by a corporation, a partnership or a group of heirs. For some operational purposes, it is more important to know the name of the mortgage holder, taxpayer or managing agent. These identities are frequently confused, ambiguous or deliberately concealed by the owner.

As a result of these and other obstacles, development of comprehensive building files which can be used for inter-agency activities has proceeded slowly. Washington, D.C., and Philadelphia are among east-coast cities which have developed such files. New York City plans to have one in use by the spring of 1972. New York's has been used on a pilot basis to prepare a list of selected office buildings for use by the Fire Department in evaluating a new fireproofing code for fully air-conditioned buildings and development work has begun to use it for estimating population.

In general, the most significant work in this area has been done in localities which grew up in the days of electronic data processing. Southern California communities have been particularly outstanding in developing and using Building Files.

Los Angeles County, for example, has had a tape-oriented property records system since 1958 for use by the Assessor, Tax Collector and County Auditor. The system contains 1,850,000 records describing every parcel in the county. Los Angeles County contains 77 cities plus countless water, school and fire districts and other taxing entities with overlapping jurisdictions such as are found in most counties. All property tax bills are calculated by the county. Receipts are distributed to the local taxing groups based on their particular tax rates and the amount of taxable property in their jurisdiction.

The Los Angeles master parcel file contains the following data: tax block and lot number, which uniquely identify the property; the address of the property; the name of up to two owners; the address of each owner if it differs from the property address; the land and building dimensions; the tax area codes which identify the agencies with jurisdiction for that parcel; zoning classification, use codes, current and prior year valuations for both land and buildings; history of up to three last sales prices for the property; homeowner, veteran and religious exemption codes. In the last two years the county has been adding more detailed data describing specifics of construction on the property; year built, quality, class and shape of building, number of bedrooms, number of bathrooms, total square feet, cost of construction, quality of view, presence of mineral deposits and other data which would help the assessor in more accurately computing the value of the property.

This data is of great value for functions such as city planning, renewal and site selection activities. Local governments are working with the county authorities to obtain greater access to this wealth of data.

Looking into the future a few years, we predict that building files will be accessed by on-line systems, with every agency that currently keeps building related files, having instant access to fields related to their special areas of interest. Thus, the Fire Department can develop a history of fires by building and even apartment; the Housing Agency can manipulate data concerned with elevator inspections, building code violations and their status by building or apartment, rent control status by apartment, building and occupancy permits status; the Environment Pollution Agency can access fields concerned with incinerators, boilers and so on.

Thus, instead of each agency developing and maintaining its own building file, it can be centrally maintained, and where required, the interagency information can be readily analyzed for operational and planning purposes by other than the agency controlling the data in the field of interest.

In addition to ACG/GBF's and Parcel/Building Files, several other files may be developed, depending on local conditions. These could include *Alias Street Names* which include alternate names or alternate spellings of street names in the locality; *Familiar Places* which includes places not known by normal address, such as parks, landmarks, department stores and shopping centers; *Highways* which identify limited-access highways in terms of segments; and *Boundary (Polygon) Files*. Boundary Files are used to describe the perimeter of a jurisdiction. They may be organized in terms of the street intersections which form the boundary of an area, or, especially if used for automatic mapping, they will give the (x, y) coordinates of the points which form the polygon describing the area.

All of these files require considerable effort to develop and maintain in urban areas which are complex and constantly changing. However, as they are used more widely, more local agencies will participate in their development and exchange of information.

ACCESSING (MATCHING) APPROACHES

ADMATCH

Admatch is a set of address matching programs that enables a blockside type of GBF (e.g., ACG, DIME, etc., file) to be matched against any data file on street address. This package was originally developed by the U. S. Bureau of the Census for the New Haven Census Use project in 1967. It has since been significantly upgraded and improved.

It consists of a preprocessor and a matcher program separated by a standard sort program. The preprocessor program analyzes the address field of each record to be matched, identifies the specific components of the address and places these components into specific fields of a record segment called the "match key" which is appended to the original data record as part of the preprocessor output record.

For example, the address:

2520 South Flatbush Avenue Extension
Brooklyn, New York

would have the address components stored in the "match key" as follows:

2520	=House number
*South	=Primary directional
Flatbush	=Street Name
*Avenue	=Primary Street Type
*Extension	=Secondary Street Type
*Brooklyn	=County
*New York	=State

All the * items are first standardized through the use of tables before being placed in the output record.

The next step in the overall matching procedure is to sort the preprocessor output file on house number within odd/even-side of the street parity code within street name (within county, within state if a multi-county and multi-state file is involved).

The sorted data file is now matched on the "match key" field against a blockside GBF reference file "match key" field that has previously been created via preprocessing and sorting as above. The matcher program compares the data record match key to all the GBF records with the same street name and selects the "best match." The "best match" is determined by a weighting scheme determined by the user. That is, each component of the address is given a weighted value and thus each component that matches, or in the case of house number, falls within the high-low blockside range of house numbers gets the weighted value added to its total weighted score. Thus, a record can be accepted as "matching" even though each address component does not match identically with the "matched" GBF record.

Admatch is available in IBM 360 DOS and OS and RCA Spector 70 versions, at a nominal fee ($60) from the Bureau of the Census. Over 100 municipalities and regional entities are using this package. The package requires a minimum of 32K bytes of magnetic core to run under DOS.

In Charlotte, North Carolina, a file of 35,000 sewer connection records were put into machine readable form and via Admatch, matched against a master water billing file and a master sewer billing file. Seventy-five percent of these records matched at a 98 percent acceptance level. The balance of the man-matched records were then manually processed. Of these, 10 percent were true non-matches, that is, they had sewer connections but were never billed for water and/or sewer services. This resulted in considerable additional yearly revenue for Charlotte.

In the Los Angeles area, both the City of Los Angeles and the Census Bureau's Southern California Regional Information Study (SCRIS) have been making extensive use of Admatch.

Building Permit information has had Census Tract and Block numbers added to it. This has been done for every year since 1960. It enables one to ascertain the yearly growth of the housing stock and via the housing stock, an estimation of the population growth.

Motor Vehicle applications have been coded with Census tracts and blocks via Admatch. This enables transportation planners to get an accurate fix as to where the vehicle owners are located.

Orange County, California has used Admatch to code housing inspection reports with census tract and block codes. This is used to analyze the quality of housing stock in the various geographic areas of the county.

UNIMATCH

The Bureau of the Census is currently developing this package which, in essence, is a generalized file linkage system. It is structurally similar to ADMATCH, however, not only will it match on street addresses, but also on street intersections, major traffic generators or any other logical connection. This generality is achieved by allowing the user to specify what fields to compare, what comparisons to make, what significance to attach to a success or failure to compare and finally what action to take depending on the level of success of that comparison.

STREET ADDRESS MATCHING SYSTEM (SAMS)

SAMS is a proprietary package of Urban Data Processing, Inc. of Cambridge, Massachusetts that matches addresses in a fashion similar to Admatch. This may not be too surprising since the authors of SAMS were involved in the development of the original Admatch package. SAMS processing, like Admatch, consists of a preprocessor run, a sort on the house number and street name, and a match against a similarly processed GBF.

SAMS operates under IBM 360 DOS and OS operating systems with a minimum of 96K bytes of core storage.

The City of New York (our home town) has used SAMS for numerous address matching applications. In one very dramatic application, we were able to match the Department of Social Services Welfare Recipient file to the GBF and affix the census tract and block number to 95 percent of the records. The Board of Education then used the file to determine the school district of each welfare recipient between the ages of 5 and 18. This was ultimately used to satisfy both the Federal Government and community school boards in the allocation of federal free lunch money to the school districts based upon the percentage of school children qualified for this program and not on a simple percentage of school children in each school district. With over 1,000,000 school children and over 300,000 of these qualified for the free lunch program via their welfare status, one can readily see the difficulty in performing this analysis any other way.

Work is under way to use SAMS to automatically code each birth and death record with the health area and district of the person's home address. This will eliminate a time consuming and somewhat inaccurate current clerical effort. Plans are also under way to use SAMS to add all the various types of geographic descriptors to the citywide building file we are creating. The building file, which contains the key dimensions and building classifications of each building in the city can be used for population estimation. Couple this with the geographic description, and one can then readily estimate population by specific census tracts, health areas, school districts, . . . etc., for a multitude of planning activities.

OTHER BATCH PROCESSING ADDRESS MATCHING PROGRAMS

Kettering, Ohio, has developed an Admatch type package that operates on an IBM 1130. They have utilized it to code and analyze health data by various geographic descriptors.

Raymond Soller of Florida State University has developed an Admatch type package for the CDC 3600 which may be available to other CDC users.

REAL TIME ADDRESS MATCHING

A number of real time address matching systems exist. They tend to differ based upon the amount of conversation one wishes to accept between the terminal operator and the system and the average amount of time, during peak load conditions, that one is willing to wait in order to find the address in question. Generally speaking, the less conversation between the system and the operator and the faster the access required, the more expensive the system in terms of programming and file storage.

In Chicago, building code violation complaints are being entered on-line via terminals. The street name is first converted to a standard street number code. The house number is then used with the street code to access the file. In New York City, the Police Department has installed a system, SPRINT, which enables telephone operators to enter the request for emergency help, including the house number or the street intersection into the system via CRT type terminals. The system analyzes the address portion of the message and finds the blockside record which either:

(a) contains the house number within its high-low address range or
(b) is adjacent to the intersection.

Finding the matching blockside record enables the system to then determine:

(1) the police precinct and sector for that blockside
(2) the police car(s) nearest the precinct and sector that are free

and to flash the original message and the police car availability message to a radio dispatcher who in turn broadcasts a dispatching message to the police car(s) he selects to assign to the call. It should be noted that the SPRINT address file was built from the same base file that was used to create New York City's GBF. This illustrates that a GBF can support both batch processing and real time operating systems.

ACCESSING OTHER GEOGRAPHIC DESCRIPTORS POINT-IN-THE-POLYGON ACCESSING

This approach requires that a file be develcped which represents, in essence, polygons for the specific geographic descriptor of interest. That is, the extreme (X, Y) coordinate sets for each vertex of an area, say a health area, are recorded as a specific polygon. This is done for every health area in the system. Then, given the (X, Y) values for a specific item of interest, say a birth, one can determine, through the use of a point-in-the-polygon algorithm which polygon, and thus which health area, the item falls in. Tulsa, Oklahoma, has used this approach in a real-time system to determine which police district a specific address lies in, so that it can dispatch the proper police car to the scene.

MULTI-INDEX GEOGRAPHIC FILES

Kettering, Ohio, has developed a multi-indexed geographic information system which enables one to access their 22,200 record parcel file by (a) plat, section, and lot code; (b) tract, block and parcel number code; or (c) book, page and index code in addition to the parcel's address. A series of index files for each of these access items has been developed to support this capability. This system has been used by their Planning Division to obtain building and land valuations on proposed zoning changes and to analyze the parcel file to determine the existence of non-conforming land uses.

COMPUTER GRAPHICS

In this section we confine the broad subject of computer graphics to the aggregation, analysis, reformatting and graphic display via computers for the support of urban planning in all its facets.

In almost all instances, government use of computer graphics has taken the form of producing maps or map representations at various scales and, most important, the insertion into these maps of the aggregation and factor analysis of data in varied formats.

To focus more sharply on our topic let us glance at some classes of data displays as they have appeared on maps used by government urban planners.

1. Specific quantitative data by volume, capacity or total incidence represented on the map by shading and/or specific numbers.
2. Exception type data resulting from the comparison of two quantities, such as surplusses or deficits of various resources.
3. Time oriented data which focus on trend analysis or the comparison of specific resources at two or more points of time.

It might be well at this point to remind ourselves that there is a considerable difference between an information system supporting an administrative process such as invoicing, payroll or tax collection, and one designed to support any planning or programming activity. In the first instance the system must handle data by individual documents or transactions. The planner, however, is concerned with data behavioral patterns, trend analysis of many types, simulations or model studies. All of these demand the aggregation of information from many diverse sources, such as the Bureau of the Census, Housing and Development Authority, or the tax assessor's records—and perhaps the total incidence of police or fire calls, welfare cases and so on.

With the increasing mountain of data being made available by the growing world of computers, the planner, to avoid being buried or immobilized, must be supported by distilled data in the form of computer reports or graphic displays or, most likely, a combination of both.

The need of a combination of maps and specific data was proven, and was used by military, navigational, and government planners thousands of years ago. Today the fantastic growth in available planning data and the volatile nature of today's world will force further sophistication and use of computer-driven graphic displays.

Now let's take a look at the classes and specific types of computer-driven graphics currently in use by government at various levels.

HIGH-SPEED PRINTERS

First and currently most widely developed and used are high-speed printers which use the characters of the

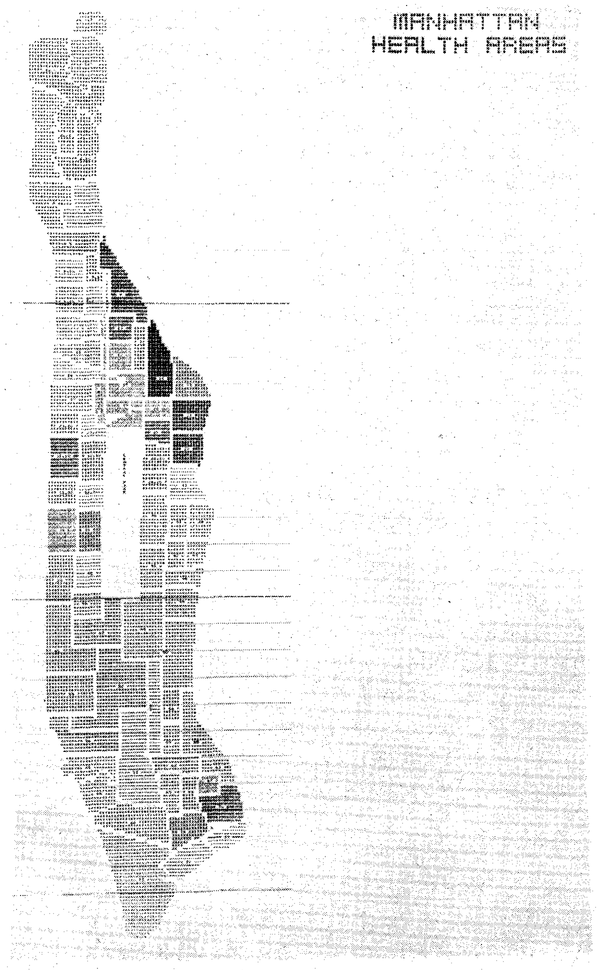


Figure 1—Manhattan Health Areas—Shading is proportional to number of public assistance cases—SYMAP—Conformal option

printer and overprints of characters to produce various levels and densities of shading to illustrate data on areas representing map areas. Since lines cannot be drawn on this type of map, boundaries of areas can only be approximated. These approximations, however, are often adequate for planning support. In addition to shading, specific data values can be shown for certain points on the map.

The two mapping programs using high-speed printers that are in most common use by governmental planners are "SYMAP" and "GRIDS."

SYMAP is the best known, most comprehensive, and most widely used computer mapping program currently available. It uses a standard high-speed printer to print typewriter-like characters, on a standard 11×15-inch computer printout sheet. Lines can be roughly approximated with printer characters, and areas can be shaded with up to 10 progressively darker shades. The darkest shades are created by overprinting two or more printer characters.

The original SYMAP program was written in 1963 under the direction of Howard Fisher at Northwestern University. The program is currently being maintained by the Harvard Laboratory for Computer Graphics, where it is supported by a technical staff, ongoing research, and teaching materials such as correspondence courses and seminars.

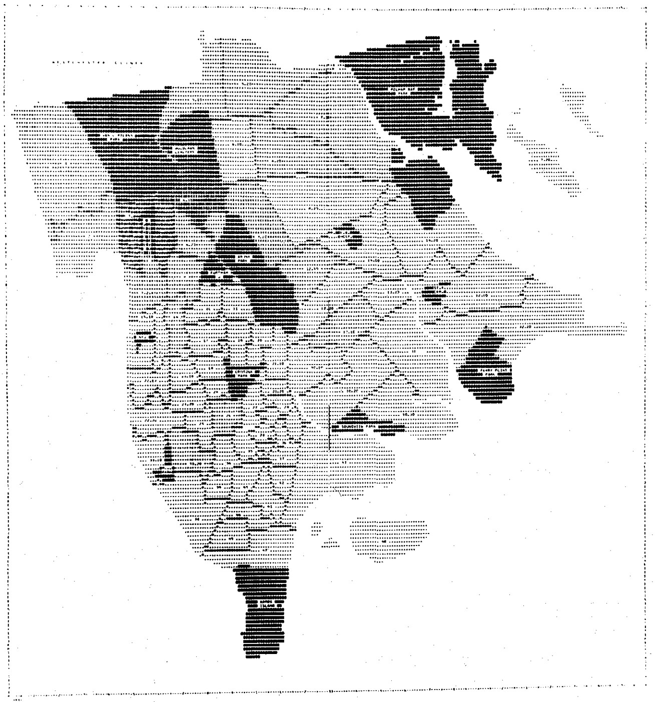


Figure 2—Outpatient residence distribution for Montefiore Hospital by Bronx health areas—SYMAP—Conformal option

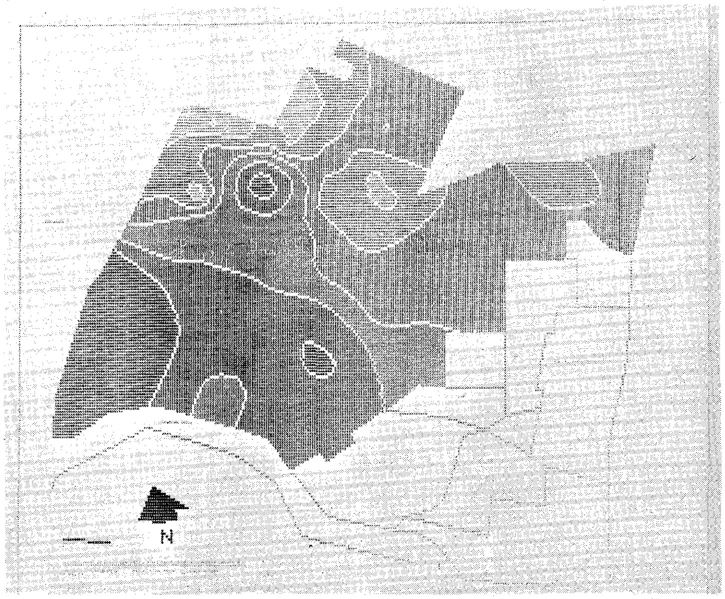


Figure 3—Ratio of shelters to daytime population in Long Island City—SYMAP—Contour option

Generally, SYMAP has been programmed for the larger scale IBM systems, although modifications of SYMAP have been used on other computer vendors' large scale computers.

SYMAP enables three basic types of maps to be produced through its three primary options: the conformal option, where the areas shaded are approximate representations of the polygonal geographic areas that they represent; the contour option, where the data values are assigned to particular points (such as block face centers or block or tract centroids) and the program shades contours by establishing equal-width intervals representing the range of values between each pair of points; and the proximal option, where the data values are again assigned to particular points and the program shades the respective values from each point to an imaginary line equidistant between each pair of adjacent points.

In Conformal (see Figures one and two), the areas shaded represent the actual areas which the data values represent. In Contour (see Figures three and four)[2], the areas shaded represent equal-width intervals between the centroids of the data areas. For example, between the centroid of the area with data value six and the centroid of the area with data value four, three equal-width intervals have been established representing data values six, five and four. In Proximal, the areas shaded are represented by imaginary polygons whose boundaries are described as equidistant from the centroids of the actual data areas. For example, the

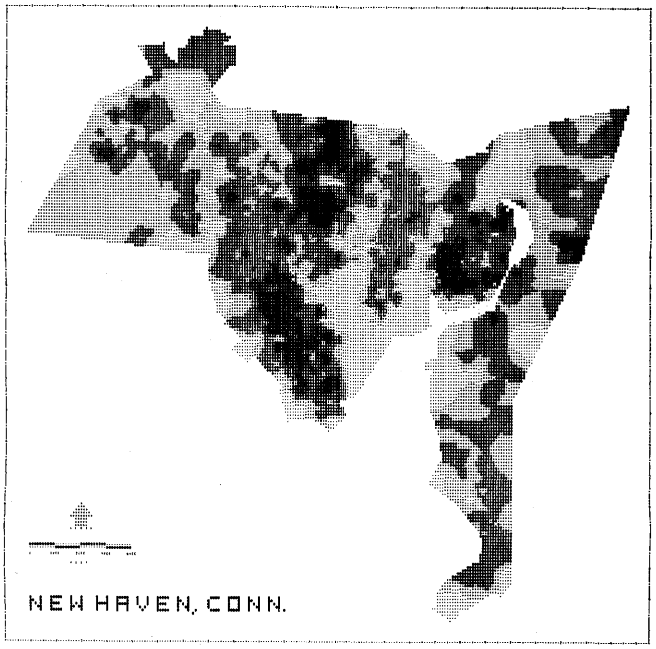


Figure 4—SYMAP shading map showing contour levels—Density of preschool children

boundaries of the imaginary polygon describing data value two are equidistant from the centroid of the actual data area for two and the centroids of the surrounding actual data areas.

In addition, SYMAP has a large number of statistical-support options which permit calculations of means, standard deviations, histograms, and percentile groups—all within the same mapping program package.

Within the SYMAP system, changing from one type of map using one mapping option to another type of map is not easy. For instance, switching from a map using irregular polygon areas, as in conformal block shading, to a map using single data points, as in contour shading, requires the input of a completely rearranged geographic data base.

"GRIDS" (Grid Related Information Display System) produces maps by dividing the area to be mapped into a network of rectangular grid cells. A grid cell may be as small as one printed character or as large as 55×55 characters.

The system was developed by the Census Bureau's Southern California Regional Information Study in an effort to simplify required programming skills and data preparation.

GRIDS is very flexible both in digesting input data and producing maps. There are three types of maps available: (1) shaded, in which data values are represented by overprinted characters of varying darkness; (2) density, in which a character is splattered randomly throughout each grid cell with more characters representing higher values; (3) value, in which the actual data value is printed in each grid cell.

GRIDS accepts as many as 10 input variables to be mapped, provides for manipulation of the data in any desired way, and produces up to five different maps for each run. No previous preparation of the data is necessary and no knowledge of the data values is necessary. GRIDS provides for data manipulation through MAPTRAN, a built-in FORTRAN-like programming language, or a user exit routine if necessary.

GRIDS is especially easy to use because all specifications are free format keyword type and there are many default values for the user with simple needs.

The program will run on nearly any system with a FORTRAN Compiler. It has been run on an IBM 360-30 with 32K bytes of storage.

Among the most published users of line printers and SYMAP for graphical display programs is the U. S. Census Bureau. They have used both census related data and locally obtainable data to perform comparisons and display of results such as Densities of Populations of Preschool Children, and Hour's Spent by Visiting Nurses (Figures four and five).[2]

The University of Kansas, at Wichita, has experimented with line printer graphics at the block level using SYMAP, to display correlated data of land value

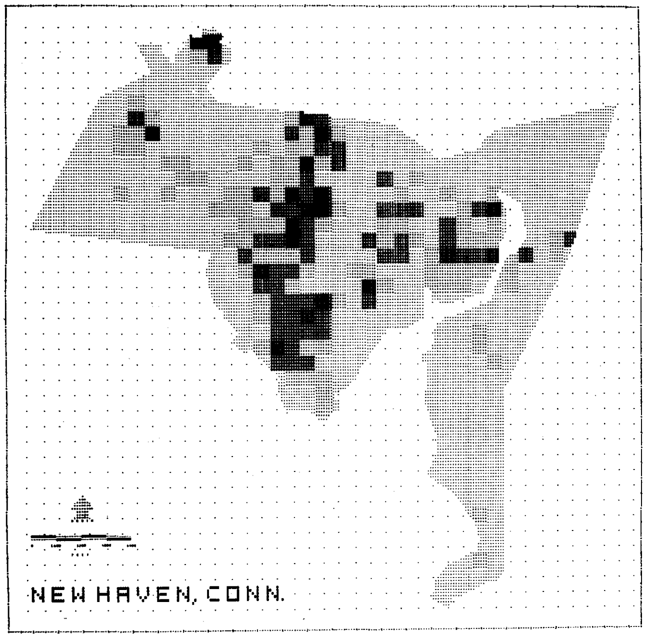


Figure 5—SYMAP shading map showing grid cell levels—Adjusted hours spent by visiting nurses association

by block area. This is different from Census Bureau studies which are predominantly oriented to aggregations at the tract level.

New York City has made extensive use of SYMAP capabilities to analyze and display a variety of City problems, including air pollution, health statistics and public assistance cases. It has also been used in conjunction with a highly sophisticated emergency preparedness display system.

The Battelle Memorial Institute is using SYMAP in conjunction with their pollution research efforts. These include studies that determine patterns of waste-water discharges from industrial and municipal operations and to evaluate their effects on surrounding waters. The research technique, developed by Battelle, consists of collecting aerial infrared and tracer dye imagery of surface water discharges. Data recorded from the infrared imager is processed and a high-speed printer then prints out isothermal plots, density plots, and contour plots. The contour plots provide two different views. Used with a stereoscope, these two views provide simulated three-dimensional temperature contours.

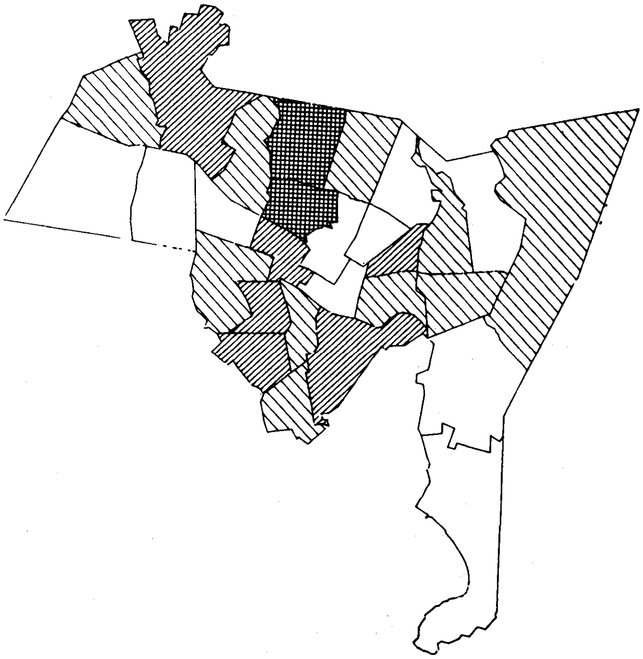


Figure 6—Pen plotter tract outline map showing tract levels—Percent non-white

PEN PLOTTERS physically move a pen over a piece of paper, under program control, to produce a graphic output. Examples of graphics produced by these plotters are shown in Figures six, seven and eight.[2] The first mechanical computer-driven pen plotters were flatbed models on which the pen is moved over a flat sheet of paper in both the X and Y direction.

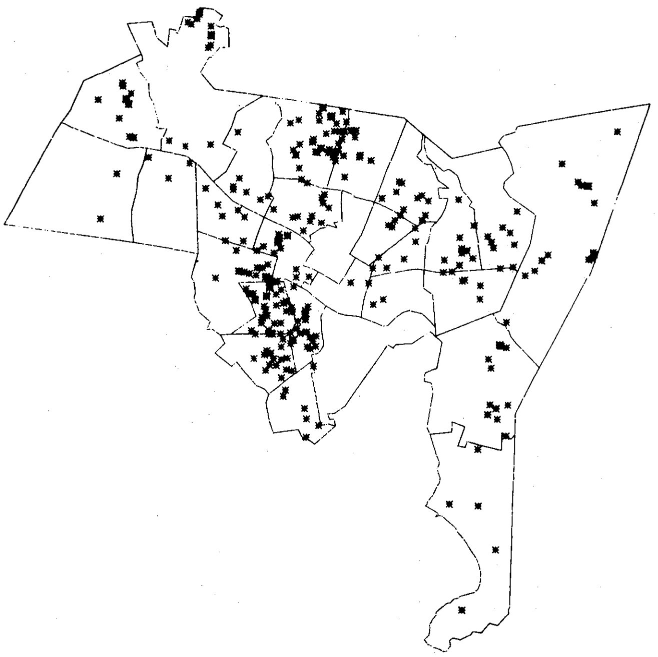


Figure 7—Pen plotter tract outline map showing incidence occurrence—Under-weight births

Later, California Computer Products, Inc. (CAL COMP) and the Benson-Lehner Corporation developed

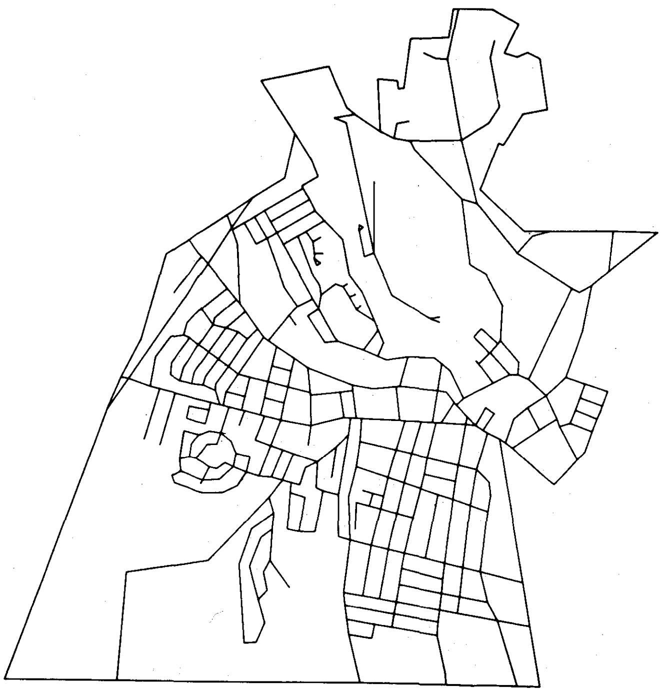


Figure 8—Pen plotter outline map of Northwestern New Haven

drum plotters which move the pen to obtain one direction and the paper to obtain the other.

An important difference in plotters is the manner in which the movement of the pen is controlled.

Analog Plotting Systems convert the (X, Y) distance through which the pen must be moved to a voltage or other electrical measurement which in turn is used to drive motors through a proportional number of turns to reposition the pen. Thus the digital input is converted to a specific distance or analog value.

Digital Incremental Plotting Systems drive the pen (and the paper, in the case of drum plotters) on a quite different principle. The drive mechanisms on these machines operate in discrete steps, or increments. The step size is usually 0.005 or 0.010 inch, or the metric equivalent to the nearest tenth of a millimeter. So-called step motors control these motions. The step motors can make from two hundred (on the slowest plotters) to about sixteen or eighteen hundred (on the fastest) steps per second for a corresponding plotting speed of between two and eighteen inches per second.

The input to the step motors comes from the magnetic tape or computer to which the plotter is attached. This input, under program control, specifies the exact number of steps through which the motors are to move. There is thus no conversion from digital to analog values. At any one time the program preparing input for the plotter "knows" the exact location of the pen, and thus, even after drawing thousands of lines on the paper the pen can return to the exact starting point.

There is another important difference between the analog and the incremental plotters. On the former the regular pen is useful for drawing lines, and can draw them quickly once set into motion; but the line-drawing pen is very slow if used to annotate the plot with numbers or letters. Hence, many analog plotters have a separate device, which may ride on the same arm as the regular pen or on a separate one and which is used exclusively for annotating the plot. The mechanism usually consists of a type wheel set in a vertical plane, which is rotated to bring each desired character next to the paper and then forced down to make its impression on the plot through a ribbon. The process is slow and cumbersome. Such a character-printing turret can increase the price of the plotter by several thousand dollars. Moreover, the size of the characters is fixed, and they can be placed on the plot only in one or at most two orientations.

On the incremental plotters the same pen mechanism that draws lines is also used to perform all desired annotation. Only with such plotters is it usually practical to do extensive annotating on a plot. The characters can be made in any size, at any desired orientation, and they may be of any kind, subject only to the symbol table built into the computer programs used to create the plot.

The New Haven Census Use study was an early user of a GBF and SYMAP to plot census data. This was done to show comparisons of data at the Census Tract level, with distinctly drawn tract boundary lines, such as percent of non-white to white populations throughout New Haven, Connecticut (Figure six).[2]

The Santa Clara County (Cal.) Planning Department has developed a census tract polygon base file which they have been using with a pen plotter to graphically depict employment, population and housing data by Census tracts within the county. They are also using their pen plotter to prepare map overlays, which

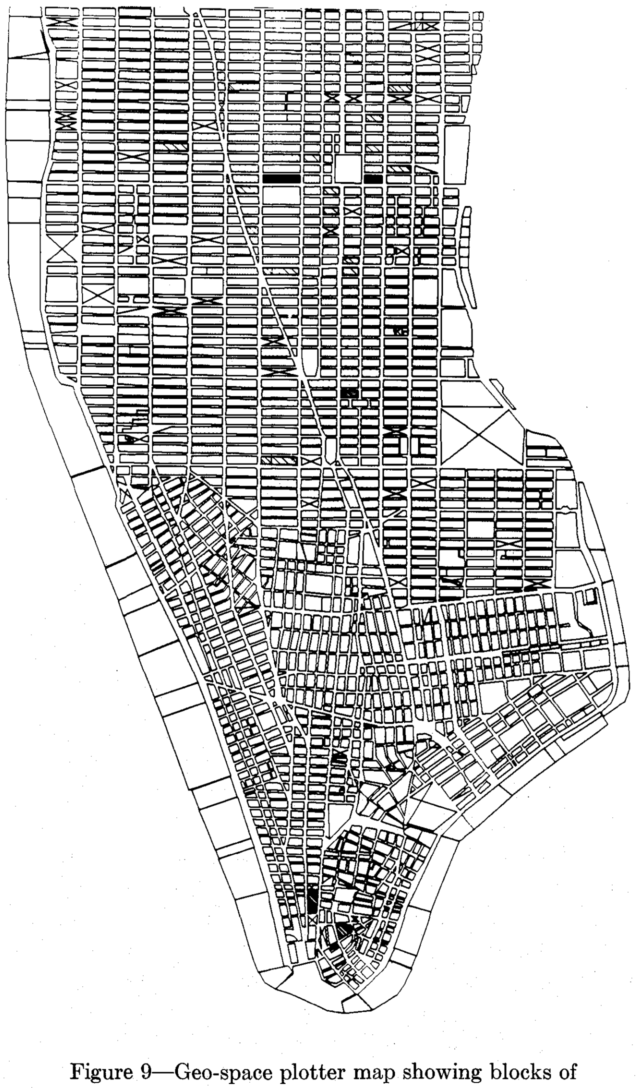


Figure 9—Geo-space plotter map showing blocks of lower Manhattan

correspond to county base maps with a high degree of resolution, to illustrate census tract level information in conjunction with block level street maps.

New York City, as part of a voter redistricting demonstration project, prepared a block map of a section of the City which indicated the number of registered voters at the approximate location of their house on the block.

CATHODE RAY TUBE (CRT) PLOTTERS most likely will show the greatest advancement in computer graphics in the next few years. As most of us know, a cathode ray tube (CRT) is a display device similar to a television picture tube for which programs are written to control the generation of displays on the face of the tube. These images can be observed, recorded on microfilm for permanent storage or recorded on photosensitive paper which can be developed into distributable hard copy.

There are a number of CRT based scanning and plotting devices currently in the research and development stage and under intensive study by various branches of the government.

Although some of these projects offer the possibility of significant advances in computer graphics, conclusions that would be useful to you do not yet exist. We will confine our discussion, therefore, to hardware and systems that have given at least limited support to government planners.

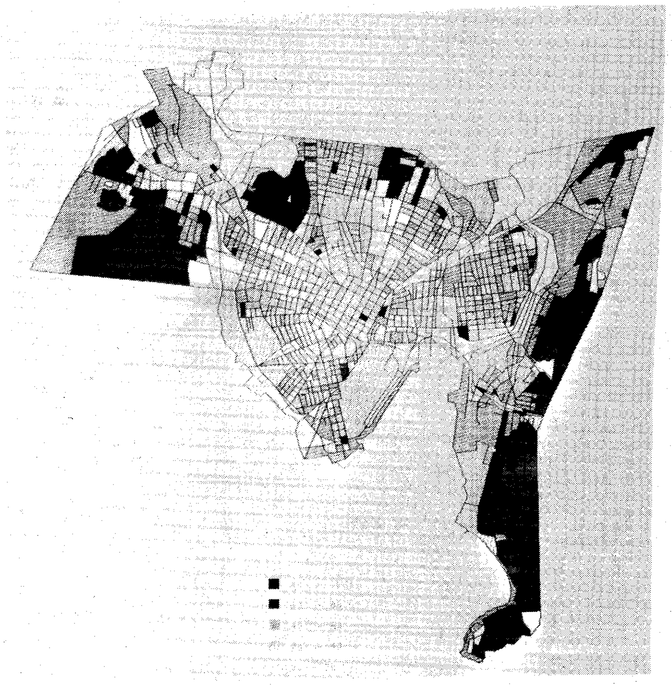


Figure 10—Geo-Space plotter shading map identifying block area levels percent of owner occupied housing

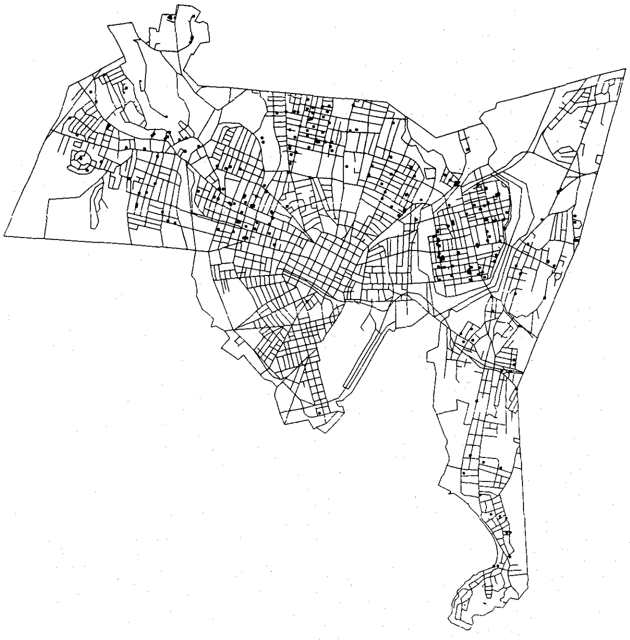


Figure 11—Geo-Space plotter outline map showing incidence occurrence—Police calls

The *Geo-Space* Plotter produced by the Geo-Space Corporation of Houston, Texas is a digital CRT drum plotter that will produce map and data representations on up to 40″×60″ film or photo sensitive paper. Figures 9, 10 and 11 illustrate maps drawn by this device (Figures 10 and 11).[2]

The Geo-Space uses a cathode ray tube and a lens assembly which focuses its ray on the film or paper. The CRT and lens assembly are housed in a tube that can be moved horizontally 40 inches. The media (paper or film) is attached to a drum large enough to hold a sheet sixty inches long. The arrangement is analogous to a drum pen plotter with the CRT and lens replacing the pen.

The Geo-Space is designed for on-line operation with a digital computer and is available with the channel interface for several different computers. Information to be plotted is developed within the digital computer central processing unit and stored in a prescribed format in core storage, or disk storage. The information to be plotted is then transferred from the on-line storage device to the plotter for display.

Plotting can be accomplished at 32 levels of intensity. A plot of 40 by 60 inches can be completed in less than two minutes regardless of the amount of data to be plotted. This rate is many, many times faster than the fastest mechanical plotters.

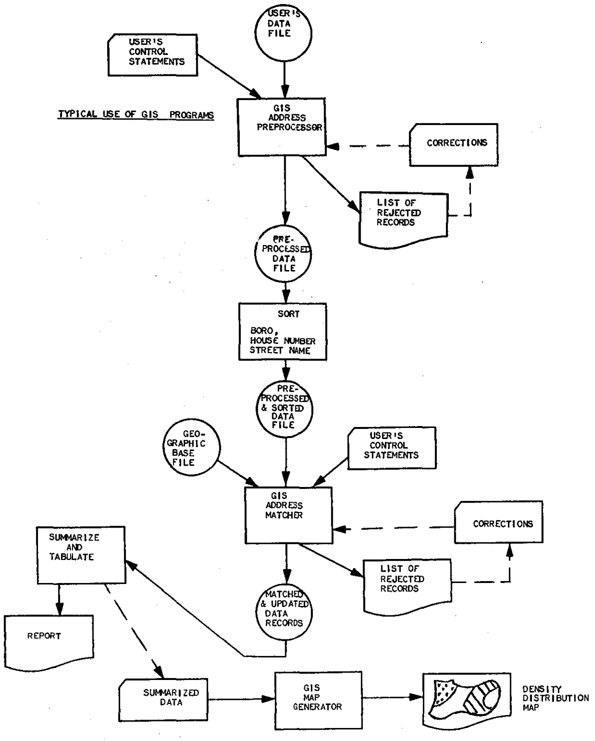


Figure 12—Typical use of GIS programs

The software support supplied by Geo-Space is a program called ALPACA which uses FORTRAN subroutines to perform character generation, generate lines and circles and control the operation of the plotter.

It might be timely to remind ourselves that each computer driven plotter, including the Geo-Space, requires the user to develop the programs needed to edit and to reformat the data to be displayed in the array or sequence demanded by the plotter under consideration.

In the case of Geo-Space, the data to be plotted must be manipulated within core on a large computer or retrieved from a disk on smaller computers to enable the plotter to produce 2-inch wide strips up to 60 inches long and then produce the adjacent 2-inch strip until the entire sheet (up to 40 inches wide) has been completed.

SCRIS (Southern California Regional Information Study) is currently developing user software to facilitate data comparisons and display of Census data on a "Geo-Space" CRT plotter. They are also preparing state and county outlines for use throughout the United States. Previous use of Geo-Space equipment was performed during the New Haven Census Use Study.

The United States Army Map Service, in conjunction with IBM, is developing software for use with IBM's Multipurpose Digitizer/Plotter. This work has been primarily concerned with automated cartography; however, the equipment and software developed appears to have direct applicability to urban data users.

A frequent need in planning operations is to determine where a pattern or a concentration of a particular activity is occurring. New York City, using a Geo-Space plotter, has prepared a block map of Manhattan, indicating where high valued real estate changed hands. This provided a good indicator of the areas where new building activity is likely to be concentrated and where new utilities and support services will be needed.

TYPICAL GIS PROCESSING

Figure 12 is a system flowchart that illustrates the typical processing one may go through in utilizing the principal components of a GIS.

SUMMARY

Geographic Information System development and application is proceeding at a rapidly accelerating pace. We envision a continuance of this over the foreseeable future as improved files get built, as more files are put on-line with more sophisticated accessing schemes, and as these systems become more and more involved with operational as well as planning type applications.

REFERENCES

1 *The DIME geocoding system*
U S Bureau of the Census Census Use Study
Washington D C 1970

2 *Computer mapping*
U S Bureau of the Census Census Use Study Report No 2
Washington D C 1969

UNIMATCH—A computer system for generalized record linkage under conditions of uncertainty

by MATTHEW A. JARO

U.S. Bureau of the Census
Washington, D.C.

INTRODUCTION

The Census Use Study of the Bureau of the Census is planning to release a computer program package to the general public providing a generalized record linkage capability, under the name of UNIMATCH (for UNIversal MATCHer).

For the purposes of this paper, *record linkage* may be defined as the process of examining each record independently on a *data file*, and comparing information with a domain of records on a second file, which will henceforth be referred to as the *reference file*, according to pre-determined comparison criteria with the object of selecting a unique record on the reference file that maximally meets or exceeds the criteria (if such a record exists), and transferring (or linking) information from the selected reference file record (if any) to the data file record.

If a unique reference file record is successfully obtained for a given data file record, that data file record is considered to be *accepted* (or matched), and the transfer of information from the reference file to the data file can occur. If no unique reference file record is obtained, the data file record in question is considered to be *rejected* (or nonmatched), and no linkage of information will transpire.

The most rudimentary example of the record linkage process is the file update problem, where a master data file is examined sequentially, and a key field (such as a record control number) is compared with an identical field on the transaction update file (the reference file). If the fields match, information is transferred from the reference file to the data file, thereby updating the master record. If the fields do not match, the data file is advanced to the next record with no data transfer occurring. The reference file is sorted by the key field, and every equal or high comparison causes the reference file to be advanced by one record. In this case, the domain of the candidate records on the reference file is one, since a single reference file record is a prospective match candidate for each data file record (it is assumed that the key numbers are unique in this example).

CRITICAL FIELDS AND CANDIDACY DOMAIN

Any key field where an exact match is necessary before any further comparisons or data transfer can be considered will be referred to as a *critical field*. It follows then, that a mis-match on any critical field is sufficient to cause the rejection of the data record. Matches (successful comparisons) on all critical fields are necessary (but often, not sufficient) conditions to cause the acceptance of a data file record.

Critical fields serve to limit the domain of reference file candidacy. If no critical fields are defined for a record linkage application, the entire reference file must be examined for each data file record. This universal domain will provide the greatest potential for a high match rate, but is often prohibitively expensive in terms of execution efficiency, and generally does not provide a sufficient cost/benefit for a real application. The total number of records that must be examined by the system in a universal domain application is the product of the number of data file records and the number of reference file records, added to the number of data file records, so that large files will require a considerable number of comparisons.

The number of records that must be examined in the other extreme, where the reference file candidacy domain is limited to one record (as in the file update example), is equal to the sum of the number of data file records and the number of reference file records.

In this case, the selectivity of the system is nearly zero, since at most only one record is considered for candidacy. For this reason, a match on critical fields only will provide the least potential for a high match rate.

The reference file domain of a real application can be determined by weighing the reliability gained by increasing the domain against the resultant cost increase. Let the *average domain* be defined as the number of reference file records accepted as match candidates for the average data record. The average domain can be computed by first summing the number of candidates for each data record. If a data record does not match on all critical fields, the number of candidates is considered to be zero. If a data record does match on all critical fields, the number of candidates is equal to the number of reference file records having identical critical field values. The final sum can be divided by the number of data records read, yielding the average domain. The average domain ranges from zero (if no data record matches any reference file record on all critical fields) to the number of reference file records (if the entire reference file becomes the domain because of the omission of critical field comparisons). The average domain is not meaningful if all comparisons are critical field comparisons (that is, there are no probabilistic or weighted comparisons), since the result of the match is based only upon the values of the critical field keys, and additional reference file records bearing the same key values need not be examined. In the file update problem, for example, where the only comparisons made are key field comparisons, no additional information will be obtained by reading all reference file records having the same key values. In fact, such applications generally require that the key numbers be unique.

THE COST OF A RECORD LINKAGE APPLICATION

The *cost* of an individual record linkage application can be computed as follows:

$$\text{Cost} = n\cdot C_n + m\cdot C_m + \max(n,m)\cdot C_c + D\cdot n\cdot C_d + f\cdot n\cdot C_t + H \quad (1)$$

where:

n = number of data file records read

m = number of reference file records read. All the reference file records on the file will not be read if the reference file contains critical field key data values that are greater than any found on the data file.

C_n = cost of reading and writing one data file record.

C_m = cost of reading one reference file record from original data set.

C_c = cost of comparing the critical fields. This cost is an average cost, since the highest level critical fie!d mis-match terminates the comparison algorithm.

D = average domain. The number of reference file records accepted as match candidates for the average data file record.

C_d = cost of comparing all non-critical fields for one data record with one reference file record within the domain.

f = the fraction of the file that will be matched (successfully linked).

C_t = cost of linking one reference file record to one data file record. This is an average cost based on the number of comparisons and lengths of strings to be moved.

H = system overhead.

The cost of comparing all non-critical fields for one data record with one reference file record within the domain (C_d), which is the most complex factor in equation (1), may be computed as follows:

$$C_d = g\cdot(C_z + P_x\cdot C_x) + \sum_{i=1}^{g}\sum_{j=1}^{k_i} C_{ij}$$

where:

g = number of ranked groups (the concept of ranked groups is discussed in the section on ranked grouping).

k_i = number of fields in group i.

C_z = cost of determining whether group score for one reference file candidate record exceeds maximum score.

C_x = cost of storing all group scores in the case that the reference file candidate record has a higher score than any yet encountered.

P_x = fraction of ranked groups whose scores are monotonically increasing (i.e., the number of groups where it is necessary to replace a previously encountered maximum score divided by the total number of groups).

C_{ij} = cost of comparing field j in group i. This includes the cost of string compression (such as SOUNDEX), field padding, fixed or interval logical comparisons, weight determination and adding the derived weights and penalties to the total ranked group scores.

Note: The derivation of these terms will become apparent when the ranked grouping algorithm is discussed.

CRITICAL FIELD HIERARCHY

Since the critical fields serve to limit the domain of reference file candidacy, each critical field is an identification key. For sequential files, these identification keys must be sort keys, and both the data file and the reference file must be sorted on all such keys in ascending collating sequence. For direct access files, the identification keys can be used to retrieve records sequentially. For the purposes of this paper, it is assumed that both files are sequent:al files, since direct access files can be processed sequentially.

Let the *major critical field* be defined as the critical field with the highest level (major) sort key. The major critical field must be the major sort key. The *minor critical field* (or *search field*) is, of course, the critical field with a sort key of lower hierarchy (more minor) than any other critical field. It is therefore the minor sort key in the set of all critical fields, and not necessarily the true minor sort key. The minor critical field is the search field. This field is used to construct the domain of reference file candidacy. All reference file records having the same search field (minor critical field) contents as the equivalent field on the data file will become prospective match candidates for that data file record. S:nce the minor critical field defines the domain (which is the most important factor to the cost of a record linkage application), this field should be chosen with great care.

THE UNIMATCH LANGUAGE

Before embarking upon a discussion of the record linkage theory governing the comparisons of noncritical fields, the relationships between critical field hierarchy, domain construction, and field definitions within the UNIMATCH language will be explained. The basic operation of the system and the structure of the UNIMATCH language are prerequisite topics toward an understanding of these relationships, and will therefore be discussed presently.

The language itself is composed of a series of statements consisting of fixed-field keywords with operands, and tables which follow some statements. The keyword identifying a certain statement type is always punched in column one of the control stream record. The number and position of the operands are dependent upon the individual statement. In general, the statements are not directly executable commands (like the FORTRAN Arithmetic statements), but are for the most part, definitions. The statements are edited, and reduced to numerical form by the UNIMATCH Compiler. Arrays defining the structure of the problem, and blocks containing control information are created. The UNIMATCH Assembler reduces this basically tabular compilation into blocks of relocatable executable machine code for the IBM System/360, and blocks of control information. These blocks are loaded and executed under control of the UNIMATCH Executer. The three programs of the UNIMATCH system (the Compiler, the Assembler and the Executer) are written in IBM System 360 Assembler Language, and both OS and DOS versions of the system will be released.

As many runs as desired may be compiled, assembled and executed by each phase, respectively, so that multiple-pass linkage applications can be executed during one job step. Splitting the system into three phases enables the generation of efficient program code that will not penalize the user for features not desired, and provides greater core storage for users of limited computer facilities. Furthermore, production runs need only be compiled and assembled initially because the assembler creates object modules that can be retained.

FIELD DEFINITION

Due to their importance in subsequent discussion, considerable attent:on will be g:ven to the KEYFIELD and EQUATE statements. In general, however, exact details of the language will not be presented in this paper. Every field used in a comparison (critical or non-critical), and every fie:d used in a field manipulation context must be defined via the KEYFIELD statement. This statement consists of the keyword (KEYFIELD) followed by the beginning character position of the field, the field width in characters, and a 1-8 character label (or field name) which provides reference to the unique field. Unless otherwise specified (via the EQUATE statement), all fields are considered to be identical (having the same beginning column and fie:d width) in both the data and reference files. The inclusion of a field in the KEYFIELD or EQUATE statements does not necessarily imply that any actions or comparisons will be made on the field, but :nclusion does define the comparison configuration, should subsequent statements require such comparisons.

The EQUATE statement allows the user to define the appropriate comparison configuration for the defined fields, and to establish which fields belong to either the data file or the reference file (the omiss'on of an EQUATE statement for a field results in that field belonging to both the data and reference files). The EQUATE statement consists of the keyword followed by a two character code, followed by two or three names (prev:ously defined in a KEYFIELD statement). The first name given in all cases is called the base name. All subsequent references to equated fields should be by base name. The true configuration of the comparison and al fields involved will be utilized in creating the object code blocks, without having to be restated by the user. A code of FC or FN requires two names: the name of the data file field followed by the name of the reference file field. The locations and lengths of these fields have been previously defined (by KEYFIELD statements). The field on the data file will be compared character by character with the equated field on the reference file should any subsequent statement require a comparison. Codes FC and FN differ in their treatment of unequal length fields (the data file field is not equal in length to the reference file field). FC causes padding of the short field with blanks to the right, and FN causes padding of the short field with leading zeros, to equalize the lengths. FC can therefore be used with character data, and FN with numeric data.

Codes RI and RP requ're three names: the name of the data file field, followed by the name of the reference file field contain:ng the low extreme of a closed interval, followed by the name of the reference file field containing the high extreme of the interval. By this means, comparisons may be made between fixed values on the data file, and closed intervals (where the endpoints are included in the interval) on the reference file. Subsequent statements will only require the base name (in this case the data file field name) to cause the appropriate interval comparisons to be made. Code RP is identical to RI except that the parity of the fixed field must match the parity of the low extreme of the interval. In this case parity refers to whether the values are even or odd. Parity agreement is useful for street address matching, where a house number is to be compared to a range of addresses on a street, and the odd and even addresses are on differing street sides. Codes DI and DP pertain to fixed fields on the reference file being compared to intervals on the data file. All interval fields are treated as numeric, and the short fields are padded with leading zeros at execution time before comparisons are made.

CRITICAL FIELD DEFINITION

The MATCH, SORT, and SEARCH statements define the hierarchy of the critical field structure and the domain construction. The keyword MATCH, followed by a field name, indicates that the field is a critical field, and the appropriate comparisons will be made for the configuration defined via the KEYFIELD and EQUATE statements. A mis-match on any critical field is sufficient to reject the data file record. The domain of candidacy is thereby reduced by means of the MATCH statements. If a critical field represents an interval comparison, the ranges covered by the interval must not overlap. The SORT statement defines the sort order of the critical fields by requiring a level number and critical field name. The SEARCH statement defines the reference file domain. The keyword *SEARCH* is followed by a field name (which must appear in the SORT statement). All reference file records having the same search field contents as the equivalent field on the data file record will become prospective match candidates for that data file record. Operationally, reference file records are read into core storage, until the end of the partition is reached, at which time they are written on disk and paged in as required. The program will operate more efficiently in larger storage partitions. If the domain is chosen carefully, there should be few (if any) input/output operations of this sort. The program will process domains as large or small as desired without any user modification. The disk paging operations are optimized to minimize the number of accesses.

WEIGHTED COMPARISONS

After the reference file domain is constructed, it is often desirable to make a number of comparisons with the object of selecting the most probable record for a match if that record is within the acceptability tolerances. The concept of *weighted* comparisons makes this possible. Much work has been done on the derivation of weighting schema. A paper written by Sidney H. Brounstein of the IBM Corp.[1] provides a comprehensive review of the weighting techniques developed, and will be quoted extensively in this section. Robert T. O'Reagan of the Census Bureau[2,3] has done considerable work in the derivation of practical weighting algorithms. The fundamental weight'ng theory underlying most modern methods is the "Direct Weighting Concept", derived by Newcombe[4,5]. The following quote from Brounstein will illustrate the basic concept:

> Given a pair of records that are candidates for a match, the number of the components that agree and disagree is observed. If equal impact were given to each component, the discriminating power of some components that are measurably

> more powerful than others would be wasted. For example, if two American health records agree on the surname Smith, that would be some ind:cation that this m'ght be a true match. However, if they agree and both say Hinton in that component, it is a much stronger clue. The rarer the argument, the less like y that accidental agreement will occur between components in records brought together at random.
>
> If the sets M (matched pairs) and U (unmatched pairs) preexisted in totality, there would be no matching problem. The problem exists because some method of characterizing M and U is required to decide to which of the sets a new comparison pair (representing two records not previously linked) belongs. This implies the use of sampling methods, computations based on frequency analyses of the separate files, or inferences based on knowledge of the total populations from which the data files are drawn.

Practical weights and penalties (negative weights) can be computed by determining the following four probabilities (by dividing the frequency of occurence by the number of comparison pairs tested).

$P_1 = Pr$(component agrees|pair is a true match)
$P_2 = Pr$(component agrees|pair is not a true match)
$P_3 = Pr$(component disagrees|pair is a true match)
$P_4 = Pr$(component disagrees|pair is not a true match)

The appropriate weight for the comparison pair can be computed by:

$$S_w = \log_2 (P_1/P_2)$$

The appropriate penalty can be computed by:

$$S_p = \log_2(P_3/P_4)$$

Weights and penalties can be derived in this manner for all components (or configurations of components). A component is a field for which a weighted comparison is to be made, and its configuration is the value of the field on both the data and reference files. The score for each prospective candidate in the domain can be computed by adding (separately) the weights and penalties for each component (determined before the actual record linkage application by means of a sample population, and the above method). The reference file record with the greatest weight and lowest penalty is the most probable match within the domain. If the penalty of this best record is greater than a certain threshold value, the data file record can be rejected on the basis that the probability of the chosen reference file record being a correct link is too low for the particular application.

This method of computing weights and penalties assumes the statistical independence of each component (if this is not true, then joint distributions must be computed by taking the set of all components at one time, generally an immense undertaking). For address matching purposes (geographic coding), the assumption of independence is fairly good. Allowing this assumption, the addition of the logarithmic weight scores for each component is equivalent to multiplying the likelihood ratios, thereby attaining the joint probability for the record being a match.

The following example will illustrate the method of deriving weights for an address coding application. The probabilities P_1 to P_4 are computed using a sample file where all records have been linked manually (assuming no clerical error!). Consider the component "street type":

$P_1 = Pr$(street type agrees|pair is a match) $= 0.64$
$P_2 = Pr$(street type agrees|pair is not a match) $= 0.16$
$P_3 = Pr$(street type disagrees|pair is a match) $= 0.02$
$P_4 = Pr$(street type disagrees|pair is not a match) $= 0.74$

These probabilities are based on frequencies of occurrences divided by the number of comparisons made. One comparison is made for every reference file record in the domain for a given data file record. The weight for a match on street type is:

$$\log_2(0.64/0.16) = \log_2(4) = 2$$

The penalty for a mis-match on street type is:

$$\log_2(0.02/0.74) = \log_2(1/32) = -5$$

Next consider an additional component "direction". Assume that weights derived in a similar manner are 4 and -6 respectively.

In future record linkage applications the two components can be examined against the equivalent reference file components. If both street type and direction matched, the total weight for the candidate record would be $2+4=6$, with a penalty of zero. If direction matched, but street type did not, the weight would be 4 and the penalty would be 5 (the penalty score is considered to be a positive number, not a negative weight). The converse case would produce a weight of 2 and penalty of 6. Disagreement on both components would produce a weight of zero and a penalty of 11.

THRESHOLD LEVELS

The we'ghts and penalties computed in the previous section can be used to determine the threshold level for the data file record rejection. If the weight obtained for a particular record is equal to the penalty score, then there is a 50 percent chance (1:1 odds) that the reference file record will provide a correct match. If the weight obtained is one greater than the penalty, the odds favoring a correct match are 2:1. If the weight is less than the penalty, the odds do not favor a correct match. In general, the difference between the weight and penalty scores is the logarithm (to the base two) of the odds favor ng a match.

Odds in favor of a correct match $=2^{(S_w-S_p)}$

The amount of error tolerable for a given application must be determined by the user, in order to set an appropriate threshold level.

RANKED GROUPING

The UNIMATCH system has a feature whereby as many fields as desired may be grouped together. Grouping establishes a separate weight and penalty score counter for the set of fields in each group, as well as group hierarchy (or ranking). The selection of the best reference file candidate is determined by examining the domain, and retaining only those reference file records having the maximum weight in the highest ranking group. All other records are removed from the domain. The remaining records are examined, and only those reference file records having the maximum weight in the second highest ranking group are retained. All others are removed from the domain. This process is continued until the lowest ranking group weights are examined. In practice, this process is conducted in a single pass through the reference file domain. The weights for all groups are computed for a given record, if the weights are better than the maximum encountered previously, then the current record becomes the prospective candidate, if not, the next record is inspected. At the termination of the scan, the weight and penalty counters reflect the values of the best record, and pointers indicate the position of that record in the domain. A record is better than a previous one if there exists a weight in a high ranking group that is greater than the existing weight (weight scores are maximized within rank, e.g., a record bearing a score of 1,4,7 will not replace a record bearing a score of 2,1,1 where the first number in each set of three is the highest rank).

A further condition is added to the criteria for reference file record selection: If the weight score for any group (regardless of rank) is zero, that record will not be considered a prospective candidate. This convention allows a secondary hierarchy to be constructed. It may be desirable to automatically reject the data file record if certain minimum information is desired.

The ranked grouping concept allows a system of intuitive weighting to be established, where the ranks are assigned upon the basis of the information content of each field, and group of fie'ds. More importantly, the ranked grouping concept permits reduction of the reference file domain when the fields cannot be expressed as sort keys (via the critical field MATCH statements). A common example of this is the comparison of a field to an interval where the intervals are allowed to overlap. No sorting scheme will be able to construct a domain that will ;nclude various values of the single field if the ranges of the intervals vary and the values overlap. Including all va'ues of the intervals (within the critical sort keys, of course) into the domain, and defining the interval comparison to be within a group, and assigning a weight of one for a match and a weight of zero for a mis-match, will result in the retention of all intervals containing the field value (or vice versa, if the interval is on the data file) in the domain. Another example, is the phonetic compression of a character string (another feature provided by UNIMATCH). All values sufficiently close to the data file value can be retained in the domain. This is useful for name matching. The interval comparison method given is useful for matching a street address to a range of addresses in a city.

IMPLEMENTATION OF WEIGHTED COMPARISONS

The UNIMATCH system requires all fields where weighted comparisons are needed, to be organized into ranked groups. The GROUP statement requires a group name (which is a 1-8 character label not previously used, that will uniquely identify the group), and a list of all fields (by field name) that comprise the group. Any particular field may be used in as many groups as desired. At least one group must be defined to allow weighted comparisons, but fields may be grouped in any fashion desired. Each group carries its own weight and penalty score counters. The LEVEL statement defines the group ranking. It consists of the level number and group name. The ACCEPT statement defines the maximum penalty score permitted for the group named. If the penalty score of the chosen candidate (the reference file record with the highest ranked positive weight scores) is greater than the amount specified in the ACCEPT statement, the data file record will be rejected.

WEIGHT statements must be provided for all fields specified in GROUP statements. Rather than specifying the two states discussed earlier (match and mis-match), the WEIGHT statement prov des for five possible states:(1) the data file field matches the reference file field, and neither field contains blank nor zero entries (match);(2) the data file field contains a blank or zero entry and the reference file field does not (data file omission);(3) the data file field contalns a coded entry and the reference file field contains a blank or zero entry (reference file omission);(4) both fields contain blank or zero entries (null match);(5) the fields conta'n conflicting non-zero (or non-blank) entries (mis-match). These states also apply to intervals.

An interval is considered null if both extremes are zero. Conditional weight tables may be created which will modify the group weights contingent upon certain configurations of reference file field values. For example, if the "street type" field is considered, and the data file record has no entry coded, it might be desirable to give added weight to those reference file candidates containing a street type of "ST", "BLVD", "RD", etc., over those containing street types of "PL", "CT", "SQ" etc., on the basis that the frequency of occurrence of "ST", "BLVD" etc., is likely to be higher. Conversely, if the data file field value is "COURT" and it matched the reference file field, a higher weight may be assigned because the frequency of occurrence is less than "ST" and therefore the information content is greater.

DATA TRANSFER CONTROL

Since the object of the record linkage process is the transfer of data from the reference file to the data file, considerable thought was given to the design of an effective data transfer control system. The system allows the user to code a number of executable commands describing the procedure for transferring information from the chosen reference file record to the data file record (or if the data file record is rejected, transferring messages to the data file record). The unconditional "move" is the simplest operation. A field in the reference file is moved to a location on the data file (or appended to the end of the record, if desired). A "compare" command causes transfer of information contingent upon the stated value being present in the data file or reference file. This allows records of different types to be processed correctly. UNIMATCH maintains a set of internal flags which indicate whether individual fields were successfully matched for the chosen candidate record. Data transfer can occur contingent upon the successful linking (or non-linking) of any individual field for the candidate.

This feature makes it possible to take alternative actions for various comparison results. A practical example is in address coding against a Geographic Base File where the records contain two address ranges, one for each side of the street. If a match occurred with a range on one side of the street, it is desirable to transfer the geographic codes relating to that side only. Other commands permit examination of the group weight and penalty scores providing for data transfer contingent upon a range of values. Character text can be moved into the data file record instead of information from the reference file record. The transfer of text can be conditional upon comparison results, weight and penalty values, etc. Simple arithmetic functions can be executed upon the weights and penalties, allowing for example, the placing of the odds favoring a match on each data file record. A memory feature allows the reformatting of fields on the data file, imputation of information on the data file, as well as computations on data file fields.

FIELD MANIPULATION AND FILE CONTROL FEATURES

The UNIMATCH system provides the capability of converting input field values to new values on the basis of conversion tables. For example, area codes may differ between both files, in which case a correspondence dictionary may be prepared to provide for conversion of the area codes. The capability of assigning default values to fields not coded is also provided. Selection of data file records for matching may be accomplished via SELECT statements. For example, it may be desired only to match records occurring in one city where a number of cities are present in the data file.

The file management system provides the capability of separating rejected output records from the accepted ones, or combining them as desired. In addition, it is possible to separate rejected records of different types, and to print listings of selected records in the output file. A post-processor feature allows the matcher to be used in non-record linkage applications where field conversion, record selection, and reject separation tasks can be performed on single data files.

STATISTICAL INFORMATION

UNIMATCH provides much statistical information regarding each run that is executed. The output data records contain twelve digit status codes prefixed to the body of the record which indicate the status of that output record. A code of "MT" indicates that the record was matched successfully, and the penalty scores of up to five high ranking groups are printed.

It is thereby possible to determine if a given data file record is an exact match, and if not, what the degree of uncertainty is. Critical field rejects are indicated by a code of "CR" followed by the name of the field that caused the mismatch. Group rejects produce a code of "GR" followed by the group name and the penalty score for that group. This gives the user an indication of how much he must "loosen the reins" to permit the record to be linked. Selection rejects and search field rejects are also flagged.

At the end of each run the number of rejects by type, field name and group name are printed. In addition, statistics are printed tabulating the number of exact matches, the number of input/output disk operations required, the number of different reference file domains used, the average domain, the effective domain (the average number of records for any domain), input and output record counts, etc. This information provides the user with a good picture of the effectiveness of the record linkage application.

CONCLUSION

This paper has attempted to present the operation of UNIMATCH by presenting both the theory underlying the development and the physical implementation of the system. All features and the exact details of the specifications could not be discussed in a paper of this scope. UNIMATCH will provide a generalized record linkage capability to IBM System/360 users, but more importantly will permit experimentation with varying linkage algorithms with resultant greater knowledge of the practical considerations of the record linkage problem. Since a large portion of the record linkage problem is record standardization, a Universal Standardizer (UNISTAND) is being written as a companion system for UNIMATCH. UNISTAND will scan free-form text under pattern recognition control in order to place recognizable components in fixed fields for matching. Procedures will be supplied to the user of both systems to process the most common problems (such as address coding, transportation coding, etc.). These procedures will allow the user to perform complex record linkage tasks without requiring the sophistication needed for weight and algorithm development. It is expected that these systems, along with supplied procedures, will provide the capability of solving most of the current record linkage problems.

BIBLIOGRAPHY

1 S H BROUNSTEIN
Data record linkage under conditions of uncertainty
Delivered at the Seventh Annual Conference of the Urban and Regional Information Systems Association Sept 6 1969 Los Angeles Calif

2 R T O'REAGAN
Address matching by computer
1968 U S Bureau of the Census unpublished

3 R T O'REAGAN
Application of information theory to bureau problems in record linking
3/8/68 U S Bureau of the Census unpublished

4 H B NEWCOMBE J M KENNEDY
Record linkage making maximum use of the discriminating power of identifying information
Communications of the Association for Computing Machinery 5 1962 pp 563-566

5 H B NEWCOMBE J M KENNEDY S J AXFORD A P JAMES
Automatic linkage of vital records
Science 130 1959 pp 954-959

New directions in legal information processing*

by R. T. CHIEN**

Massachusetts Institute of Technology
Cambridge, Massachusetts

P. B. MAGGS and F. A. STAHL

University of Illinois at Urbana-Champaign
Urbana, Illinois

INTRODUCTION

Present areas of application of computers to the law fall into three broad categories: (1) those involving applications of business accounting techniques such as in tax preparation and client billing, (2) those involving data management techniques such as law enforcement, criminal justice, and keyword legal source material information systems, and (3) those involving on-line file manipulation in such areas as text-editing and drafting. These systems demonstrate that computers can work very well with problems that can be expressed in terms of numbers or information that can be handled on the basis of its external form.

During the coming decade, we foresee the continued expansion of the use of existing systems and also the development of some new systems based on conventional data processing techniques. Such new systems may be expected in automating the accession to index systems patterned on existing manual systems, processing land use documents, supervising paroled offenders, etc. However, ordinary data processing methods are by their very nature incapable of providing much-needed assistance in the central area of legal work, where the content of the material at hand (laws, contracts, depositions, etc.) must be understood and analyzed in terms of its meaning and logical relationships. It is impossible, using file management techniques to deal with, for example, a set of conflicting laws or regulations. Problems such as these must be attacked at a level where the interrelationships among the various words and phrases (as well as their individual meanings) are understood.

Comprehensive machine-readable data bases of legal materials are rapidly becoming available as a result of the economies they afford in such areas as typesetting and legislative drafting. However, the conversion of legal materials to machine-readable form has not made them significantly more accessible to those who must use them. Because of the great complexity and volume of legal materials their analysis now requires great investment of time by highly trained personnel. If computers could "understand" legal materials, their attractive assets (their large reliable memory, and their extraordinary capacity for rapid and accurate information processing) would make them ideal assistants in the field of law.

The ability to provide such an understanding is emerging in new techniques currently being developed in a computer area called "artificial intelligence" in dealing with such problems as automated logical deduction, problem solving, and natural language understanding. These techniques appear to be the key to solving many of the problems of legal automation.

In particular, we foresee the development of a number of computer based services in this new technology. These include:

1. Automated systems to provide quick and inexpensive assistance for many non-controversial, but not necessarily simple legal questions to aid lawyers, social and welfare workers, administrators, police, and of course the public itself;
2. Automated consistency-checking, and consequence-finding systems to aid in codification

* The work presented here was performed at the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign. It was supported by the Joint Services Electronics Program (U.S. Army, U.S. Navy, U.S. Air Force) under Contract No. DAAB-07-67-C-0199.

** On leave from the Coordinated Science Laboratory and the Department of Electrical Engineering at the University of Illinois at Urbana-Champaign.

and law reform for legislative and administrative bodies;

3. Automated systems to assist in teaching law and legal reasoning for those who need to know the law; and
4. Automated interviewing systems for initial client and witness screening.

In this paper we include a brief survey of recent applications of computers to the law, a discussion of the further types of automation that are needed in the law, an outline of current developments in artificial intelligence which could be applied to aid in the automation of the law, and finally a description of the new directions we envision for legal information processing during the 1970s.

A SURVEY OF RECENT APPLICATIONS OF COMPUTERS TO THE LAW

Automation has been applied to problems in criminal justice and law enforcement as well as a host of other applications including legal information retrieval, income tax preparation, and legislative drafting systems. In this section a brief description of these application areas is given. For a more thorough survey of this work see Robins;[47] Bigelow,[7] the May, 1971 issue of *Law Library Journal*, which is devoted to computers and law; and the individual references cited in this paper.

Criminal justice and law enforcement information systems

For our purposes, we can view criminal justice and law enforcement information systems as consisting of users, a computer, and an ever changing data base being used to collect, store, and update information in such a way so as to facilitate the retrieval and exchange of criminal justice and law enforcement information for governmental units. The most advanced of these systems include: Project SEARCH[61] (System for Electronic Analysis and Retrieval of Criminal Histories) developed jointly by LEAA (Law Enforcement Assistance Administration) of the Justice Department and a number of participating states, the NCIC[19,23] (National Criminal Information Center) computer network of the Federal Bureau of Investigation, and the NYSIIS[22] (New York State Identification and Intelligence System).

Legal information retrieval systems

The volume of jurisprudential reference material in the form of cases, statutes, and administrative rulings and regulations has prompted the development of automated law retrieval systems to aid the legal researcher. This work has been based in part upon research being carried out in the area of library automation and is specifically related to the problem of document retrieval.

Of the experimental projects that have been undertaken for performing automated legal information retrieval the basic assumptions have been that a person seeking technical legal information can be led to relevant information using a keyword type approach, and that technical legal information can be categorized in such a fashion as to be retrieved by a keyword type approach. Within this framework there are basically two approaches that have been taken: those systems which rely on prior manual abstracting of library material and those which operate on full natural text without abstracting.

Systems requiring manual abstracting

The first system for computer storage and retrieval of legal material developed by Morgan[42,58] was used for the retrieval of case law. It employed an approach where concepts were identified and assigned code numbers which were in turn linked back to the original case. Thus, having once determined the relevant concepts one could retrieve the citations to pertinent cases. This approach was very similar to the manual indices that have widespread use today such as the West Key Number System. Other attempts at automated legal research using this approach have been made by the Federal Trade Commission,[2] and the Antitrust Division of the Department of Justice.[47]

Systems operating on full natural text

A significant departure from the manual abstracting approach is the Key Word in Combination approach of John Horty.[28,29,31] Here, no abstracting or indexing of the material was done manually. Retrieval was done by finding combinations of keywords in the original text. A researcher could specify lists of words that must appear in the same sentence or statute as well as the desired word ordering. The LITE (Legal Information Through Electronics) Project developed by the Air Force Accounting and Finance Center[37] and the Root Index System developed by the Southwestern Legal Foundation[43,49,56] are two other systems that are similar in design with the Key Word in Combination approach. The OBAR System developed by the Ohio Bar Association[45] and Mead Data Central has added significant on-line interactive capabilities to this type of approach.

The ABF-IBM project

The ABF-IBM (American Bar Foundation and International Business Machines) Project[15–18] attempted to overcome the disadvantages of either of the other previous approaches by automatically generating frequencies for each word used in the original case. Deviations or "skewness" of certain words from a normal distribution were assumed to convey information about the contents of the case.

Other applications of computers to the law

There are many other applications within the legal area for which automation has been proposed, and in some cases even implemented.[6,34,35,39] Most of this work represents straightforward applications of current technology to the particular problems encountered in the legal area using methodology well established in other areas, such as office management and business accounting. Automated legislative drafting and revision, will drafting, as well as general law office document drafting are good examples of the extension of available hardware and software that has been made available in recent years. Automated court management, legislative reapportionment, law office management, income tax preparation, land title recording, and estate planning are examples of established data processing techniques that have been used for legal problems.

THE NEED FOR NEW DIRECTIONS

The quantity and complexity of legal materials is increasing at a rate that cannot be adequately handled by traditional means. In many areas the growing demands of modern society have been met by automation. However, modern computer techniques have had only a minor effect on the three major areas of legal work where they might seem applicable: legal research, legislation, and legal education. As mentioned above numerous attempts have been made to apply scientific and business data processing techniques to such legal problems, but all have fallen short of major impact because they have been by their nature unable to deal with the verbal and logical complexities of the law.

Most areas of legal work have remained relatively untouched by automation. The problem is not that the legal profession has neglected automation; rather it is that technology has failed to meet the demands of the law. What is lacking is the capacity to give legal personnel the same sort of assistance in their routine work with words and concepts that engineers get from the computer in their routine work with numbers and formulas. What is needed is the automation of the process involved in routine legal reasoning and research.

Modern statutes and administrative regulations are of such immense complexity as to be impossible to understand for the layman and difficult even for the lawyer. The problem is particularly acute in many areas where the task of interpreting and enforcing the law has traditionally been left to non-lawyers, for instance, welfare administration.

To give an example of the problem: At present when local administrative authorities receive an application for welfare benefits, the eligibility of the applicant and the benefits to which he is entitled are determined by a large and conflicting body of local, state, and federal welfare laws and regulations which may total hundreds, if not thousands, of pages in length. Typically an applicant may have to wait for weeks or even months while an overburdened administrator attempts to determine his legal status. As another example, a business transaction involving questions of tax laws, zoning regulations, and building codes may fall through because of the delays and expense involved in the drafting of the transaction to conform to the many, and possibly conflicting, requirements of the law.

The successful development of automated techniques for dealing with complex legal situations would help in these and many other areas by providing quick and inexpensive answers to many routine legal questions. Such techniques could also help employers and employees determine their rights under labor relations contracts. They could help both the taxpayer and the government in tax planning and administration. It would be possible to provide legal guidance for some small businessman's transactions which involve too little money to justify full scale analysis by a lawyer.

In many areas, where the law now consists of confusing and conflicting regulations at different levels—federal, state, and local, the first step toward law reform would be the unscrambling of a situation that has been caused by years of haphazard legislation so as to provide a clearer assignment of rights, duties, and administrative responsibilities. However, legislators have been unable to take even this first step, not merely because of the cost, but also because human ability to enact complex legislation appears to exceed human ability to reorganize, recodify and simplify that legislation.

The total length of the United States Code and the collected statutes of the various states has doubled in

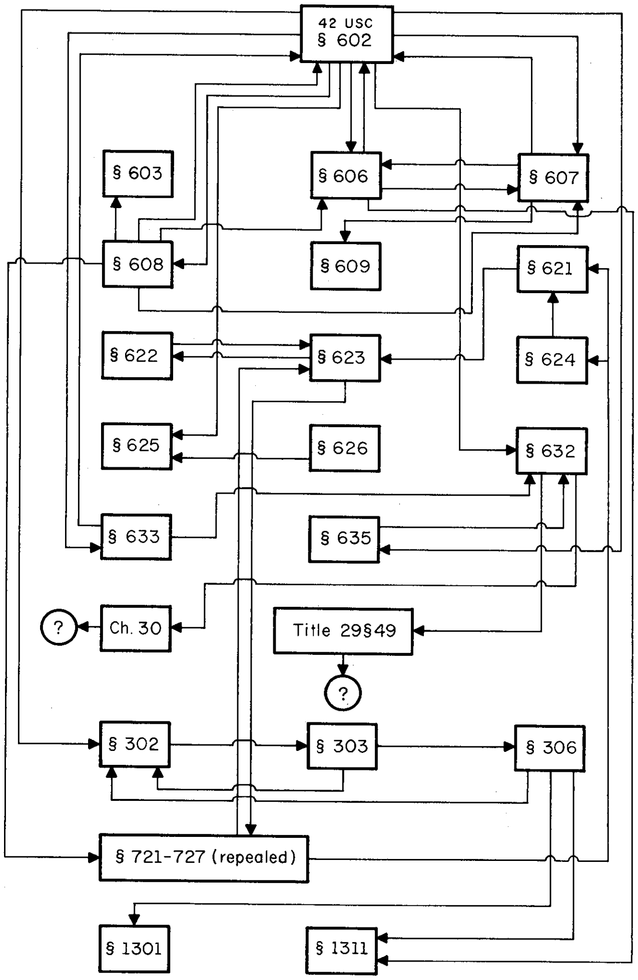


Figure 1—Network of References from 42 U.S.C. §602, "State Plans for Aid and Services to Needy Families with Children." References to the laws of the fifty states, implicit cross references, and references to federal and state administrative and judicial interpretation of the statute have been omitted for lack of space.

the past few years and promises to double again in this decade. In commercially important areas such as tax law, it is possible, at high cost, to find specialists who understand even the present incredibly complex legal picture. In many socially important areas, such as welfare law, pollution control law, environmental quality law, and urban planning law, it is almost impossible to find anyone who has a thorough working knowledge of the field. As a result, many important social programs are either slowed down or stifled.

To illustrate the complexity of the problems involved consider a single section of federal welfare legislation, for instance, that dealing with state plans for aid to needy families with children (42 U.S.C. §602). This single section refers to eight other sections, most of which in turn refer to still other sections creating an incredibly complex network (as illustrated in Figure 1). Even one of the shortest sections referred to by §602, namely §625, which defines "child welfare service" has an extraordinarily complicated internal logical structure (as illustrated in Figure 2).

There is a major unfilled demand for legal services in the United States. It is particularly acute for lower

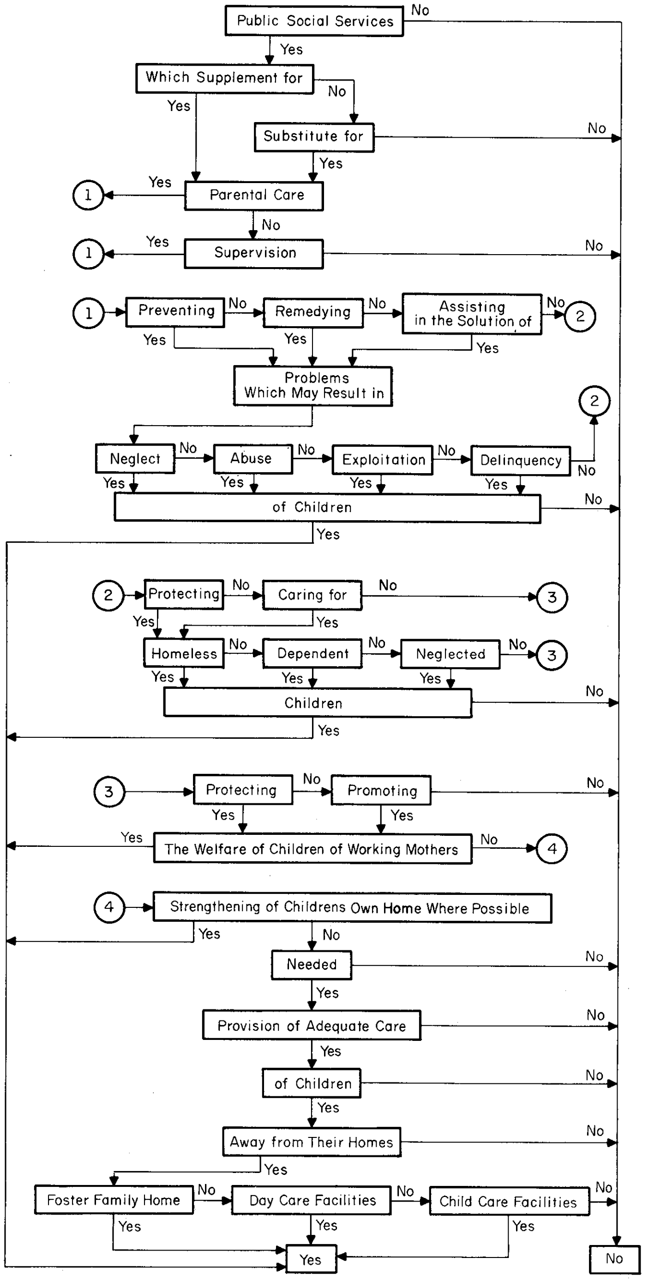


Figure 2—Logical structure of 42 U.S.C. §625 " 'Child-Welfare Services' Defined." One of the Simpler Sections. The appendix contains the text of this section

income groups, but legal services, like other unautomated services, are increasingly becoming priced out of the reach of larger and larger segments of the American public. At present large law firms provide excellent legal services for those able to pay their fees. Most of the lawyers in these firms spend most of their time in legal research and drafting legal documents.[55] The small businessman or average citizen, if he can afford legal service at all, typically employs a sole practitioner who cannot afford the library or the time necessary to do an adequate job of legal research on his client's problems.[11]

The experience of the medical profession in dealing with the problem of the cost of doctors' services suggests that two approaches be tried simultaneously. First, the rapid automation of those areas suitable for automation, and second the training of paraprofessionals to free the professional from routine tasks. However, the training of paraprofessionals in law—welfare administrators, social workers, lay magistrates, law clerks, etc., has been neglected by American education until very recently. Application of traditional methods of legal education to the problem would be possible, but it would be expensive, and would be difficult in view of the fact that all American law schools are filled to capacity.[5]

RECENT DEVELOPMENTS IN ARTIFICIAL INTELLIGENCE APPLICABLE TO LEGAL AUTOMATION

A significantly broad theoretical framework has been established for the application of computers to the law as the result of research and development in the area of artificial intelligence, in particular in question-answering systems where work has been carried on for over a decade. For information concerning such systems we refer the reader to Simmons.[52,53] Of the more advanced question-answering systems Green and Raphael[24,25] have developed a very powerful deductive procedure, and Simmons, Burger, and Schwarcz[51,54] introduced an extremely attractive representation scheme, while Biss, Chien and Stahl[8] have developed the R2 system which incorporates a number of advanced features not found in these other systems. These systems have attacked the problem of understanding facts and answering questions about them on an automatic basis. It is understood as a result of the development of these systems, how to, for example, resolve certain kinds of semantic and syntactic ambiguities. In addition, a number of sophisticated formal internal structures have been developed that allow much of the expressiveness found in natural English and yet can be manipulated by machines in a systematic fashion.

Transformation of natural English into a formal internal representation

Generally, natural language question-answering systems use some formal internal representation for information in order to facilitate deductive manipulations. In a number of earlier systems the representation was based upon some type of limited relational calculi, as for example Raphael's SIR,[64] and Black's SQA.[9] Green and Raphael[24,25] subsequently developed a system that offered the full expressiveness of the first-order predicate calculus for the representation of natural language information. Simmons, Burger, and Schwarcz[51,54] developed a system that used nested binary relations for the formal representation of natural language information. The R2 question-answering system developed by Biss, Chien, and Stahl[8] uses a high-order formal language for the internal representation of information. This system represents relations between relations and quantification of variables ranging over rather complex structures. The data base chosen to demonstrate the capabilities of this system is an informal description of the motor vechicle laws of Illinois.

Logical deduction as performed by computers

What types of "reasoning" or logical operations can be performed by a computer upon factual information expressed in a formal language? There has been significant progress in recent years in the research of deductive reasoning as performed by computers. Nevertheless the computer's ability to reason with formal concepts such as those as found in legal materials is still far from fully realized.

Automated deduction procedures have been developed for the propositional calculus, but the limitations of this calculus for most practical applications led to the search for automated procedures for the first-order predicate calculus. For the first-order predicate calculus, an effective deduction procedure based upon an automatic theorem-proving algorithm was first described by Robinson[48] and was improved upon by Wos, et al.[62–65] and others.[3,4,33] Currently work is being done by a number of researchers to find effective procedures for high-order logic where concepts that cannot be handled adequately in the first-order logic can be accommodated.

NEW DIRECTIONS

The use of computers as an aid to the legal process is already extensive as indicated by the work described above. There is, however, a vast difference between providing automated criminal justice and law enforcement information systems or legal information retrieval systems on the one hand, and improving the availability and quality of legal services on the other hand. In the first case the computer is used to accomplish well-defined but time-consuming routine tasks; the second case has been almost completely ignored with respect to the cognitive potential that computers can provide. Research in this country in this second area is virtually non-existent.

We envision the initiation of research programs within the near future to meet the need for automation in the law with the following objectives: first, to increase the ability of lawyers, administrators and the public to deal effectively with many of their legal problems; second, to aid legislative and administrative agencies in the reform and development of legislation and regulations concerning current social problems; third, to provide legal training on an automated basis for lawyers, social workers, police, and others who need to know the law for the performance of their duties; and fourth, to automate client and witness interviewing and screening.

As outlined above, the theoretical framework for such research must be drawn from the field of artificial intelligence and in particular from current investigations in automated natural language question-answering and logical deduction. Eventually the development of automated inductive logic may also offer significant aid in legal work. For an excellent discussion of this possibility see Reference 10. Numerous data bases of machine-readable legal materials already exist. The specific tasks that must be accomplished are:

1. The development of techniques for the transformation of natural language legal information into formal internal representations.
2. The development of techniques for automated logical deduction from legal materials in formal internal representations.
3. The development of practical automated legal question-answering systems based upon these logical deduction techniques.
4. The development of systems for machine assisted consistency-checking and consequence-finding to aid in the normalization, integration, and modification of legislation and administrative regulations.
5. The development of computer based legal education systems based upon the automation of the traditional Socratic method of legal instruction, using dynamic question-generating techniques rather than preprogrammed instruction; and
6. The development of conversational computer techniques for obtaining information from clients and witnesses.

The data base

A large quantity of federal and state legislation has already been converted to machine-readable form, and in a number of jurisdictions new legislation is being recorded initially in machine-readable form. The availability of legislation in this form has led to significant economies in the process of inserting amendments both at the original enactment of a statute and as a result of later legislative action, and has helped in the development of keyword document retrieval schemes. The Illinois Legislative Reference Bureau, for instance, has an advanced on-line system developed by Data Retrieval Corporation, with a dozen terminals for use in legislative drafting, law revision and keyword retrieval. While these techniques are not in the central area of legal work, their existence, and the benefits and economies they provide, guarantee the availability of data bases in machine readable form.

Existing machine-readable materials include all types of primary legal sources. These are: comprehensive codes such as the Uniform Commercial Code and the Internal Revenue Code; uncodified legislation such as state statute collections; and judicial and administrative decisions.

Selection of relevant materials from the data base

The selection of relevant materials from the data base may be based on a combination of four methods. The keyword method, despite the shortcomings which have prevented its widespread adoption in legal research, is very efficient for certain types of questions. The automation of the present manual system involving networks of case and statutory citations would present few technical problems.[36] The automation of the current system of abstracts and digests combined with new techniques of automated abstracting might save the current system from the strain of the legal information explosion. Finally, as computer time becomes cheaper and cheaper, the artificial intelligence techniques discussed below (which might be expected to bring significantly better results but to use substantially more computer time) might supplant the other methods.

Transformation of legal English into a formal internal representation

Lawyers have already developed a highly formalized language as a means of communication. Development of effective legal information systems will require investigation of the problems associated with transforming this subset of natural English into an internal representation suitable for use in legal question-answering, consistency-checking, consequence-finding, and instruction systems. This research can draw upon recent advances in automated syntactic and semantic analysis, e.g.[7,50,59]

Logical deduction

In order to automate the processes involved in legal reasoning an analysis of the logical structure of legal information is necessary. Knowledge of this structure will aid in the selection and development of effective automated legal reasoning techniques based upon theorem-proving techniques mentioned above. Within the framework provided by automated theorem-proving techniques it is necessary to find reasonable models for deductive legal reasoning. These procedures can be incorporated in the legal question-answering, consistency-checking, consequence-finding, and instruction systems described in the next few paragraphs.

Legal question-answering

Interactive legal question-answering systems with internal representation and logical deduction techniques adapted from prior research on general natural language question-answering systems to reflect the nature of legal materials should be developed, for only natural language question-answering systems allow the user to engage a machine in a meaningful dialogue in order to specify his needs and receive appropriate responses. Since almost no legal personnel have any knowledge of computer programming, communications between the user and the machine should be performed entirely in the natural language question-answering mode in English.

Consistency-checking and consequence-finding

In addition to the question-answering function of the future systems, the ability to find relevant consequences for given sets of facts with respect to specified rules, regulations, laws, statutes, contracts, etc., should be incorporated. They should be capable of determining the relative consistency of any body of legal information. At present these functions, as carried out by lawyers on a manual basis, are very time consuming and are subject to error.

These systems when completed will involve direct benefits for legislators and administrative rule-makers. They will be able to use their capabilities to help in the detection and elimination of contradictions and ambiguities in existing law. Legislators will also be able to examine more fully the implications of proposed additions to the existing body of legislations.

Automated legal instruction

Legal instruction now takes many forms.[5] Some of the methods which are and will be most important in the future lend themselves to automation using artificial intelligence techniques. These include basic instruction in legal doctrine and method designed to give the student basic skills in legal vocabulary and reasoning;[5, pp. 17–18] extensive instruction designed to expose the student to surveys of broad areas of the law,[5, pp. 21–22] and simulated clinical instruction designed to develop practical skills and awareness.[5, pp. 41–42]

The accepted method of teaching basic skills in legal vocabulary and reasoning in the United States has long involved the asking of specific questions and their answering by the students as a major if not the major component of the instructional process. However, because this procedure is effective only when applied by highly qualified instructors to relatively small groups, it has been expensive to use in law schools and it has found comparatively little use in programs of legal education for non-lawyers. An authoritative recent study places the cost per legal instructor in programs for non-lawyers at $55,000, a large figure, even though less than the $80,000 a year cost for each professor in regular law school instruction.[5, pp. 69] It is hoped that the automation of this process will reduce its cost and extend its availability.

A legal question-answering system may be modified to form a system where questions are generated automatically to aid in the instruction of the law. Such a system could supplement some of the instruction now being given in law schools, but perhaps a more important application would be in training administrative officials, police, social workers, and others whose job requires day-to-day legal interpretation and administration.

One approach to such a system would be as follows: A series of typical factual situations would be posed for each of the conditions of a given statute. In order to ask a question an appropriate selection of a number of conditions would be automatically made such that the conclusion under consideration would be true. A question could be made false by eliminating some necessary conditions, and could be made more difficult by adding some irrelevant condition. In either case the answer given would be checked for consistency against the given situation and pertinent statutes. The coverage and difficulty of the questions could be structured by a dynamic teaching strategy determined by the nature and quality of student response.

Survey courses for practitioners, advanced law students, and non-lawyers are now taught largely by a lecture method, a method adopted largely for reasons of speed and economy. Automation would allow the institution of more efficient interactive methods of this type of instruction.

Computer-simulated clinical instruction, even without natural language capability, has already proved to be of considerable significance in medical teaching. It could be of similar importance in law if artificial intelligence techniques capable of handling the highly verbal nature of the lawyer-client interview were developed. Prototypes of this approach already existing in the medical area suggest the direction simulated legal clinic instruction can take.[12,13,60]

Automated client and witness screening

An experimental program has already been put into use for automated client interviewing at a legal aid office. Because the program lacks natural language capabilities, it can only accept yes/no or multiple choice answers from the interviewee.[38] Addition of natural language interactive capability would greatly enhance the power of such computerized interviewing and allow a real dialogue between the interviewee and the system.

CONCLUSION

The American public is becoming increasingly dissatisfied with the expense, delay, and inefficiency of the legal system. Application of existing computer techniques can help with some of these problems. However, real progress during the next decade demands new directions. Recent success in the field of artificial intelligence points the way for the future of legal automation.

ACKNOWLEDGMENTS

The authors, while taking full responsibility for all errors and omissions, wish to thank Professor Stephen Marx of the School of Law of the State University of New York at Buffalo and Professor Layman Allen of the University of Michigan Law School for their many comments and suggestions related to the subject of this paper. In particular, we would like to acknowledge Stephen Marx for his suggestion on simulated clinical instruction and Layman Allen for his corrections to the logic of Figure 2.

REFERENCES

1 L ALLEN R BROOKS P JAMES
Storage and retrieval of legal information: Possibilities of automation
Modern Uses of Logic in Law 1960 p 68

2 American Bar Association Special Committee on Electronic Data Retrieval
Law/fact retrieval at F.T.C.
Modern Uses of Logic in Law 1963 p 43

3 R ANDERSON W W BLEDSOE
A linear format for resolution with merging and a new technique for establishing completeness
Journal of the ACM Vol 17 No 3 July 1970 pp 525-534

4 P B ANDREWS
Resolution and merging
Journal of the ACM Vol 15 No 3 July 1968 pp 367-381

5 Association of American Law Schools Curriculum Study Project Committee
Training for the public professions of the law: 1971
In Association of American Law Schools 1971
Annual Meeting Proceedings Part 1
Washington DC 1971

6 R P BIGELOW (ed.)
Computers and the law: An introductory handbook
(2nd edition) Commerce Clearing House 1969

7 R P BIGELOW
How lawyers can use computers to practice law—Now
Unpublished Revision of an Article by the Same Title in Law Office Economics and Management Manual
R P Bigelow ed

8 K O BISS R T CHIEN F A STAHL
R2—A natural language question-answering system
In Proceedings of the AFIPS 1971 Spring Joint Computer Conference Vol 38 AFIPS Press
Montvale New Jersey 1971 pp 303-308

9 F S BLACK
A deductive question-answering system
In Semantic Information Processing M Minsky (ed.)
MIT Press Cambridge Mass 1968 pp 354-402

10 B G BUCHANAN T E HEADRICK
Some speculation about artificial intelligence and legal reasoning
Stanford Law Review Vol 23 1970 p 40

11 J E CARLIN
Lawyers on their own
Rutgers University Press New Brunswick New Jersey 1962

12 K M COLBY D C SMITH
Dialogues between humans and an artificial belief system
In Proceedings of the International Joint Conference on Artificial Intelligence D E Walker and L H Norton (eds) Washington DC May 1969 pp 319-324

13 K M COLBY S WEBER F D HILF
Artificial paranoia
Artificial Intelligence Vol 2 No 1 Spring 1971 pp 1-25

14 J M CONVEY
Information retrieval in law: Problems and progress with legal computers
Dickinson Law Review Vol 67 1963 pp 353-362

15 W B DENNIS
Status of American Bar Foundation research on automatic indexing-searching computer system
Modern Uses of Logic in Law 1965 p 131

16 S F ELDRIDGE
The American Bar Foundation project
Modern Uses of Logic in Law 1965 p 129

17 S F ELDRIDGE W B DENNIS
The computer as a tool for legal research
Law and Contemporary Problems Vol 28 1963 pp 78-99

18 S F ELDRIDGE W B DENNIS
Report of status of the joint American Bar Foundation—IBM study of electronic methods applied to legal information retrieval
Modern Uses of Logic in Law 1963 p 27

19 Federal Bureau of Investigation
The FBI's computer network
Datamation June 1970 pp 146-151

20 R FREED
Prepare now for machine-assisted legal research
American Bar Association Journal 1961 p 764

21 S E FURTH
Automated information retrieval—A state of the art report
Modern Uses of Logic in Law 1965 p 189

22 R R GALLATI
State criminal justice information systems—NYSIIS case study
Proceedings of the AFIPS 1971 Fall Joint Computer Conference Vol 39 AFIPS Press Montvale New Jersey 1971 pp 303-308

23 F P GRAHAM
FBI operating computerized criminal data book
New York Times Nov 30 1971 p 30

24 C C GREEN
Theorem-proving by resolution as a basis for question-answering systems
Machine Intelligence 4 B Metltzer and D Michie (eds) Edinburgh Univ Press Edinburgh 1969 pp 151-170

25 C C GREEN B RAPHAEL
The use of theorem-proving techniques in question-answering systems
Proceedings of the ACM National Conference 1968 pp 169-181

26 W HARRINGTON H D WILSON R W BENNETT
The Mead data central system of computerized legal research
Law Library Journal Vol 64 No 2 May 1971 pp 184-189

27 P HOFFMAN
Evaluating legal research services
In Computers and the Law; an Introductory Handbook R P Bigelow (ed) 1969 p 51

28 J F HORTY
The 'Keywords in combination' approach
Modern Uses of Logic in Law 1962 p 54

29 J F HORTY
Use of the computer in statutory research and the legislative process
In Computers and the Law; an Introductory Handbook R P Bigelow (ed) 1969 2nd edition p 46-51

30 I KAYTON
Retrieving case law by computer: Fact, fiction and future
George Washington University Law Review Vol 35 1966 p 1

31 W KEHL J HORTY C BACON D MITCHELL
An information retrieval language for legal studies
Communications of the ACM Vol 4 1961 p 380

32 K E LATTA
Information retrieval: An automated system for case law
Canadian Bar Journal Vol 10 1967

33 D W LOVELAND
A linear format for resolution
University of Pittsburgh Dept of Computer Science 1968

34 J C LYONS
New frontiers of the legal technique
Modern Uses of Logic in Law 1962 p 256

35 J C LYONS
Computers in legislative drafting: An aid or a menace
American Bar Association Journal Vol 51 1965 p 591

36 S M MARX
Citation networks in the law
Jurimetrics Journal Vol 10 No 4 June 1970 pp 121-137

37 W McCARTHY
LITE (Legal information through electronics)—A progress report
Law Library Journal Vol 64 No 2 May 1971 pp 193-197

38 R W McCOY
University of Wisconsin computer assisted legal services project
Law and Computer Technology Vol 3 Nos 7-8 1970 pp 187-188

39 R W McCOY W A CHATTERTON
Computer-assisted legal services
Law and Computer Technology Vol 1 1968 p 2

40 J S MELTON R C BENSING
Searching legal literature electronically: Results of a test program
Minnesota Law Review Vol 45 1960 p 229

41 J S MELTON
The 'semantic coded abstract' approach
Modern Uses of Logic in Law 1962 p 48

42 R T MORGAN
The 'point of law' approach
Modern Uses of Logic in Law 1962 pp 44-48

43 *Note, science-computer—The use of data processing in legal research*
Michigan Law Review Vol 65 1967 p 987

44 T PLOWDEN-WARDLAW
Computer-aided legal information retrieval
Forum Vol 4 1969 p 286

45 J PRESTON
OBAR and Mead data central system
Law Library Journal Vol 64 No 2 May 1971 pp 190-192

46 B RAPHAEL
A computer program for semantic information retrieval
In Semantic Information Processing M Minsky (ed) MIT Press Cambridge Mass 1968

47 W RONALD ROBINS
Automated legal information retrieval
Houston Law Review Vol 5 No 4 1968 pp 691-716

48 J A ROBINSON
A machine-oriented logic based on the resolution principle
Journal of the ACM Vol 12 No 1 January 1965 pp 23-41

49 H K SCHREUR
Dual-inverted file method for computer retrieval under study at southwestern legal center
Modern Uses of Logic in Law 1963 p 162

50 J A SCHULTZ W T BIELBY
An algorithm for the syntactic analysis in the R2 information system
Coordinated Science Laboratory University of Illinois Urbana Illinois 1970

51 R M SCHWARCZ J F BURGER R F SIMMONS
A deductive question-answer for natural language inference
Communications of the ACM Vol 13 No 3 March 1970 pp 167-183

52 R F SIMMONS
Answering English questions by computer:A survey
Communications of the ACM Vol 8 No 1 January 1965 pp 53-69

53 R F SIMMONS
Natural language question-answering systems: 1969
Communications of the ACM Vol 13 No 1 January 1970 pp 15-36

54 R F SIMMONS J F BURGER R M SCHWARCZ
A computational model of verbal understanding
Proceedings of the Fall Joint Computer Conference Spartan Books 1968 pp 441-456

55 E O SMIGEL
The Wall Street lawyer
Indiana University Press Bloomington Indiana 1969

56 R A WILSON
Computer retrieval of case law
Southwestern Law Journal Vol 16 1962 p 409

57 R A WILSON
Case law searching by machine
In Computers and the Law: an Introductory Handbook R P Bigelow (ed) 2nd edition 1969 p 55

58 R A WILSON
Memorial resolution to Robert T. Morgan
Modern Uses of Logic in Law 1962 p 267

59 T WINOGRAD
Procedures as a representation for data in a computer program for understanding natural language
Report MAC TR-84 MIT Cambridge Mass Feb 1971

60 J WEIZENBAUM
ELIZA—A computer program for the study of natural language communication between man and machine
Communications of the ACM Vol 9 No 1 January 1966 pp 36-45

61 P K WORMELI
The SEARCH for automated justice
Datamation Vol 17 No 12 June 15 1971 pp 32-36

62 L WOS G A ROBINSON D F CARSON
Some theorem-proving strategies and their implementation
Argonne National Laboratory Technical Memorandum No 72 Argonne Illinois 1964

63 L WOS G A ROBINSON D F CARSON
The unit preference strategy in theorem-proving
AFIPS 26 Proceedings of the Fall Joint Computer Conference Spartan Books 1964 pp 615-621

64 L WOS G A ROBINSON D F CARSON
Efficiency and completeness of the set of support strategy in theorem-proving
Journal of the ACM Vol 12 No 4 October 1965 pp 536-541

65 L WOS G A ROBINSON D F CARSON L SHALLA
The concept of demodulation in theorem-proving
Journal of the ACM Vol 14 No 4 October 1967 pp 698-709

APPENDIX

Text of 42 U.S.C. §625—"Child-welfare services" defined

§625. "Child-welfare services" defined.

For pruposes of this subchapter, the term "child-welfare services" means public social services which supplement, or substitute for, parental care and supervision for the purpose of (1) preventing or remedying, or assisting in the solution of problems which may result in, the neglect, abuse, exploitation, or deliquency of children, (2) protecting and caring for homeless, dependent, or neglected children, (3) protecting and promoting the welfare of children of working mothers, and (4) otherwise protecting and promoting the welfare of children, *including* the strengthening of their own homes where possible or, where needed, the provision of adequate care of children away from their homes in foster family homes or day-care or other child-care facilities. (Aug. 14, 1935, ch. 531, title IV, §425, as added Jan. 2, 1968, Pub.L.90-248, title II, §250(c), 81 Stat.914.)

HOMLIST—A computerized real estate information retrieval system

by DONALD J. SIMON and BARRY L. BATEMAN

University of Southwestern Louisiana
Lafayette, Louisiana

INTRODUCTION

An automated information retrieval system for real estate dealers would be desirable when the number of business transactions is large and the file of available homes is also quite large. A manual search of such a large file may be tedious and, most important, may fail to retrieve the desired information when it is actually contained within the file.[1]

In designing such a retrieval system for high-volume dealers, the fact that many homes have quite similar specifications must be borne in mind. The designer of the system must strive for a workable balance between lack of specificity and too great a degree of specificity in creating records for such real estate files. Too great a degree of specificity will result in a very limited number of records from which to choose and too little specificity may result in an unwieldy list which could only confuse the customer. It is apparent that an information retrieval system designed for this purpose should allow for future modification so as to incorporate the optimum degree of specificity in record and file structure as determined through usage.

THE HOMLST SYSTEM

The HOMLST system is an information retrieval system designed for the recording, updating, and listing of homes for sale by a real estate dealer.

System requirements

The HOMLST system, written in the COBOL language, was designed to be implemented on the RCA Spectra 70/46-G TSOS. The M70-590 disk storage unit is required for file storage and file access is achieved through a remote terminal.

Outline of the system

The HOMLST system is designed to be operated from a remote terminal[2] and employs the indexed-sequential file processing facility[3] to add, delete, and retrieve records. These records each contain a unique record-key, part of which is constructed manually, and part by programming.

Two indexed-sequential disk files are used in the operation of the HOMLST system. One file, (HOMFIL), contains records of the homes for sale and the other, (KEYFIL), is a file of sequential numbers used to create part of the record-key for each record on HOMFIL.

The system consists of two main programs each having several routines which perform various functions. One of the main programs, HOMUP, performs updating functions and the other, HOMRET, is used for retrieval of records from HOMFIL.

Record descriptions

The record description for a record in HOMFIL is as follows:

> HOMFIL utilizes a variable length record which is accessed by a unique record-key containing 14 digits. The first 9 digit field is referred to as the HOME-RECORD-CODE and is constructed as shown below:

DIGITS	
1-2	determine price range of home—for example 15 would be entered for a home costing between $15,000 and $20,000; 20 for a home costing between $20,000 and $25,000, etc.

3	Location: North = 1, South = 2, East = 3, West = 4
4	Exterior: Brick = 1, Wood = 2, Frame = 3
5	Baths: 1 Bath = 1, 1½ Bath = 2, 2 Baths = 3
6	Carpet: None = 0, Carpeting = 1
7	Bedrooms: 1 Bedroom = 1, 2 Bedrooms = 2, 3 Bedrooms = 3
8	Carport: None = 0, Single = 1, Double = 2
9	Garage: None = 0, Single = 1, Double = 2

The remaining 5 digits are taken from KEYFIL which contains sequentially generated, unique numbers so as to make each record-key unique even in cases where the home's specifications are the same as another's and the first 9 digits are identical. The following example of a record-key illustrates how it was constructed:

15121322000150

This record-key represents a home costing between $15,000 and $20,000, located on the north side of the city, with brick exterior, having *two* bathrooms, carpeting, and *three* bedrooms with a double carport. This was the 150th home record added to the file.

The following record description, using COBOL field designators, shows the positions of the data fields within the record:

```
01  HOM-REC.
    02  DUM  PICTURE X.
    02  FIX-PRT.
        03  HOM-KEY   PICTURE X (14).
        03  ACT-PRC   PICTURE 9(6) v99.
        03  IRC      PICTURE S99
                         COMPUTATIONAL
        02  VAR-PRT  OCCURS 10 TIMES
                          DEPENDING ON
                          IRC.
        03  DETAL  PICTURE X (65).
```

DUM—this is a dummy character which, by its position in the record, is used for deletion of records. In COBOL, a READ followed by a move of HIGH-VALUE to this data field with a subsequent REWRITE will delete the desired record.

FIX-PRT—this part of the record consits of fixed length data fields.

HOM-KEY—this field contains the record-key.

ACT-PRC—this field contains the actual price of the home.

IRC—this field, depending on its value, will determine how many lines of descriptive data about the home are contained in the record.

VAR-PRT—this part of the record is variable in length.

DETAL—this field is used to enter descriptive data about the home. In the record description shown above, there may be as many as 10 lines of descriptive data each containing 65 characters.

The record description for a record in KEYFIL is as follows:

```
01  KEY-REC.
    02  DUM2   PICTURE X.
    02  KEY-NO   PICTURE X (5).
```

DUM2—this data field is used to delete the record from the file in the manner described above.

KEY-NO—this is the record-key and also the record itself. These records are 5 digit numbers, sequentially generated for subsequent use in the formation of a unique record-key for records on HOMFIL.

File construction and maintenance

The files comprising the HOMLST system are constructed as follows:

KEYFIL—this indexed-sequential file, containing only 5 digit unique numbers as records, is constructed using the PRIMARY LOAD MODE of the indexed-sequential file processing facility. A pre-determined number of records is placed on the file during this load routine.

HOMFIL—this indexed-sequential file containing the records of homes for sale is constructed by entering a previously determined HOME-RECORD-CODE on a remote terminal during the execution of the updating program, HOMUP. The program then obtains a 5 digit number from KEYFIL and deletes this number from the file in order that no other record on HOMFIL may contain this number. This 5 digit number is then appended to the 9 digit HOME-RECORD-CODE thereby comprising a 14 digit, unique record-key. The actual price is then entered and finally from 1 to 10 lines of descriptive data about the home is entered as part of the record.

The two above files are updated each time a home is added to or deleted from HOMFIL. Detailed descriptions of the updating routines are contained in the section on HOMUP's routines.

Homret's routines

Operating instructions for this program and for HOMUP are provided to an operator upon request. Both programs provide the operator of the system with simple and direct instructions in response to the message: DO YOU WISH INSTRUCTIONS (YES OR NO)?—

Other than providing operating instructions, HOMRET has essentially only one routine whose function is to retrieve records of homes upon request. In response to the message: ENTER HOME-RECORD-CODE, a previously constructed HOME-RECORD-CODE is entered from the terminal. The 14 digit FULL-RECORD-KEY is displayed as the first item in the home record followed by the actual price and the descriptive data for the home. If there are no homes matching the specifications denoted by the HOME-RECORD-CODE, a message is displayed stating this. After all homes fitting the desired specifications are listed, a message is displayed on the terminal indicating such. The operator is then asked if he wishes to enter other specifications or terminate the program.

HOMRET has an error recovery routine to provide for the entrance of a non-numeric HOME-RECORD-CODE wherein the operator is informed of this fact and asked to reenter the number.

Homup's routines

HOMUP contains a CONTROL ROUTINE from which the operator may branch to any one of three separate routines to perform various functions. These routines are as follows:

1. ADD ROUTINE—the functions of this routine were described under HOMFIL in the section on File Construction and Maintenance.
2. DELETE ROUTINE—this routine provides for the deletion of records from HOMFIL. The record-key is entered from the terminal in response to the message: ENTER FULL-RECORD-KEY. The record-key is obtained from the listing of the home record by HOMRET. It is the first item listed in the home record. When this record-key is entered and a READ to HOMFIL is issued, a move of HIGH-VALUE to the first character in the record (the dummy character) with a subsequent REWRITE, results in the deletion of the desired record from HOMFIL. As part of the delete operation, the 5 digit suffix of the record-key is replaced on KEYFIL for reuse at a later time.
3. UPDATE ROUTINE—if a change in the selling price of a home is required, this may be accomplished in this routine. The record-key is entered from the terminal and the record is retrieved indicating the old price. The new price is then entered and a REWRITE is issued thereby updating the record.

When the functions of the above routines are complete, a branch to the CONTROL ROUTINE may be executed by the operator or he may terminate the program.

There are several error recovery provisions in the above routines. If a non-numeric HOME-RECORD-CODE or FULL-RECORD-KEY is entered, the operator is notified of this and asked to reenter the number. If the actual price of the home entered is non-numeric the same messages are displayed. Also, when the price entered exceeds the range implied by the first two digits of the HOME-RECORD-CODE, a message is displayed stating this and the operator is asked to reenter the price.

PROPOSED EXTENSIONS

Since many home specifications are sometimes quite similar, a greater degree of specificity may be required in constructing the HOME-RECORD-CODE. This would require changing the length of this subfield from 9 digits to the desired length. This could be accomplished with little difficulty if the record-key length would be made greater by increasing the length of the suffix from 5 digits to possibly 10 digits. Hence, when a lengthening of the HOME-RECORD-CODE is desired, the suffix could be shortened thereby not affecting the record-key length and the total record length would also be unaffected.

EVALUATION

Retrieval of records using the HOMLST system is sufficiently fast and the record provision of descriptive data concerning the homes simulates the detailed description found in most home listings. This provides the customer with most of the detailed information he requires. This provision also imposes little format restriction when entering data for a record.

SELECTED BIBLIOGRAPHY

1 C T MEADOW
The analysis of information systems
New York John Wiley & Sons Inc 1967

2 *Spectra 70 time sharing operating system (TSOS)*
Data Management Systems Reference Manual RCA Corp
Camden NJ June 1970

3 *Spectra 70 all systems cobol reference manual*
RCA Corp NJ June 1970

Organization of a natural resources data bank system

by ALVIN J. SURKAN

University of Nebraska
Lincoln, Nebraska

INTRODUCTION

The growing dependence on electronic computers for analysis of geophysical and other environmental data, brings an increasing awareness that the organization of these data in storage and retrieval systems is intimately connected with the acquisition systems and analytic transformations the data are to undergo. Current computer technology is almost totally dominated by serial storage and processing mechanisms. This peculiarity of available hardware dictates a design of data structures oriented toward the storage, retrieval and analysis of time series.

Nearly every acquisition system, whether it consists of automatic analog or digital recording instruments or humans manually keeping records, generates time series that are subsequently analyzed as continuous sequences. The subsequent analysis requires either internal comparisons within sequences or intercomparisons among other sequences collected in the same general area. Consequently, an appropriate organization for such sequenced data should readily allow a natural decomposition of the entire data base into time series-like sequences of elements which are the values of either time varying data at fixed locations or of data differing at various geographic locations and changing negligibly during the period in which observations over a set of locations are made.

The organization of the blocks of time series data generated by some acquisition system is relatively straightforward since the acquisition device dictates the sequencing. Attention must be given only to devising an indexing system that allows each such series to be located whenever its space and time coordinates fall inside a particular search window supplied to the retrieval system.

The procedure for casting the geophysical data specifying geographic variations into a time series form is not so straightforward. For example, it may be necessary to store as a time series such irregular entities as mapped areas of rock or soil type and land use variations or descriptions of river systems and the locations of natural boundaries of watersheds, lakes and coastlines or sometimes artificial administrative boundaries created by government agencies. After it is clear how such data might be cast into a time series form, it is possible to describe general algorithms for performing operations on the series to accomplish the transformation needed in analyzing the internal structure and interrelationships of such geographic entities.

If one is to accept that nearly all geophysical data can be stored and processed in time series form, it is necessary to show how a data bank system storing the various series can be related to the acquisition system and logically organized to now accommodate current types of data and ideally, other data arising in the future. The approach taken here for the design of such a data bank system is an evolutionary one as indicated in Figure 1. The results of this paper include a review of relateds ystems, a formulation of objectives and the identification of an approach to the decomposition of the data base.

CONSTRAINTS ON THE ORGANIZATION OF THE DATA BANK SYSTEM

The system must provide a data organization flexible enough to accommodate geophysical, environmental or administrative information that is now useful in connection with the natural resources of a state. Emphasis must be placed on preserving the natural groupings arising from proximity or other regional factors which influence the sequences of acquisition or use, such as the topologic connections within hydrologic networks, geographic location within basin boundaries or position with respect to a natural divide. The same

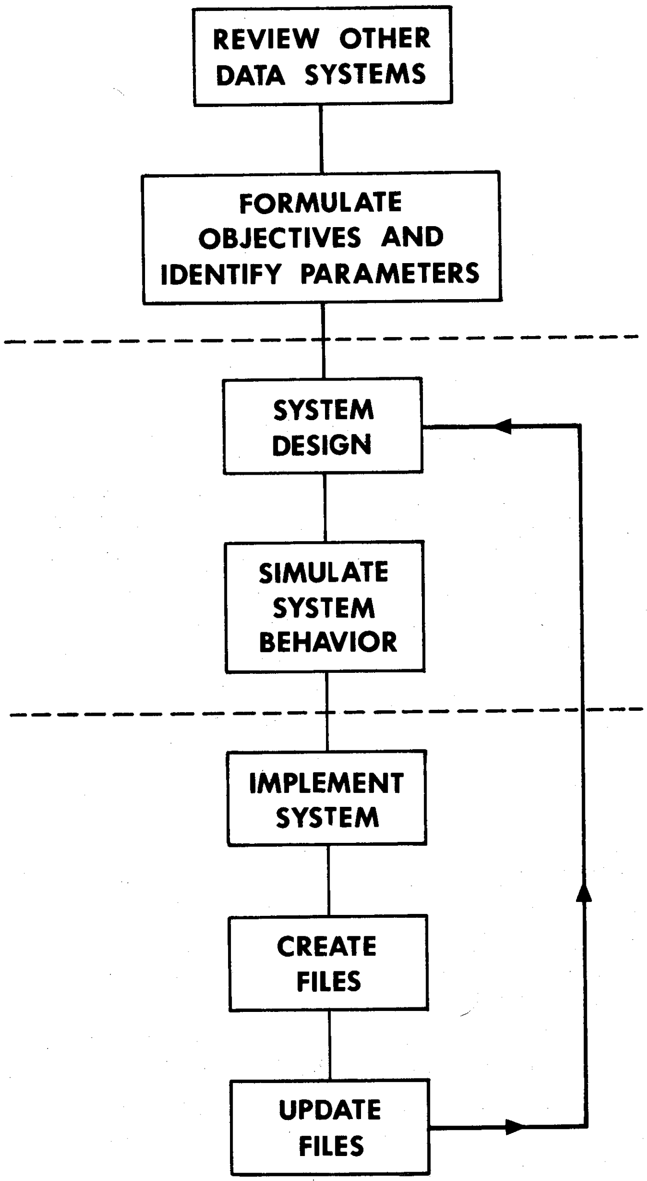


Figure 1

system should accommodate artificially grouped data within the same framework as those data with natural groupings. Artificially grouped data might, for example, be based on arbitrary and idealized administrative factors such as the positions of state or county boundaries or arbitrarily mapped survey grids, shorelines, stream limits and ridges.

Various networks such as the mathematical tree structure representations of drainage systems or administrative hierarchies, must be stored in some representation which can be easily retrieved, updated or analyzed. The analysis of such tree structures must provide the capability of identifying those subsets of structures modeled by mathematical trees which influence a particular point in the modeled system or those which are influenced by changes at a given point. For example, one might program the computer to indicate which points of a system are possibly responsible for pollution appearing at a specific point, or in the case of a known pollution source, it might be necessary to know what points could be influenced.

PARAMETERS CHARACTERIZING A DATA BANK

In describing a data bank system, it is convenient to separate the parameters into two classes. The first class characterizes the input and storage while the second characterizes the output functions and internal transformations that are associated with its operation. Parameters identified in each of these classes are summarized in Table I.

After identifying these parameters, one then seeks independent methods for finding what values apply to a data bank system needed for the geophysical data of a state. The data members will, in general, be labeled time series-like vectors. The amount of data in each consists of bit or character estimates for the size of the label plus the total number of bits or characters required for representing all the elements of the associated data vector. The total number of such vectors will, of course, depend on how finely divided, on the average, is the area of interest. For an area corresponding to an average sized state, one expects 10^5 square units of area could have varying data vectors of importance. The amount of data in each vector will typically be a few hundreds of characters or thousands of bits. Thus, the most conservative estimates of the amount of data that must be stored is 10^8 bits. In practice one must expect about 10^{10} bits and at most

TABLE I

CLASSES	DESCRIPTION OF PARAMETER	Estimate Range of Values
INPUT & STORAGE	Initial amounts of data	10^8-10^{12} bits
	Rate of adding data	10^7-10^{11} bits/yr
	Rate of data purging	0-10^7 bits/yr
	Rate of updating of data value	10^6-10^{10} bits/yr
	Levels of data categorization	6 levels
OUTPUT & DISPLAY	Rate of arrival of search requests	0.1-1.10 min
	Number of search priority categories	6 levels
	Documents retrieval rate	.01-.1 document/minute
	Number of types of document display	2-6 types

10^{12} bits after continued data accumulation concurrent with a maximum of purging of obsolete data.

The rate at which data will be added will increase with time and therefore might best be estimated for a particular time from the number of sources of data multiplied by the average number of bits or pages delivered by each. Again a conservative estimate might be 500 sources, each delivering 2×10^5 bits. These would yield 10^7 bits per year or a growth factor of 0.1 in static or permanent data. This growth factor is also reasonable for the continuously recorded time series data, if it is assumed that effectively such data would also have a ten year life and could be automatically purged to microfilm archives after that period of time.

The purge rate will then also be a function of time which is initially zero and then should grow to approach the addition rate. Such growth will require a period of time approximately of the order of the mean useful lifetime for all non-permanent information.

The rate of modification or updating of values for permanent data is estimated not to exceed 0.01 of the static component of the data base. Thus, a time of 100 years would have elapsed before the entire base would be renewed.

The number of levels of categorization of data could be kept small. The categorized data members would be vectors of typically 10^4 bits, the elements of each being addressable only by index. This would mean the number of logical routes or decision points leading to all data vectors or stored series would be between 10^4 and 10^6. Suppose there were on the average a 10-way branch or an organization tree having an average dispersion or bifurcation ratio of 10, only $\log_{10}10^4 = 4$ or at most $\log_{10}10^6 = 6$ levels of classification would suffice. An estimate of the number of levels of categorization needed is important because it is directly related to the ease with which time series information can be classified and retrieved. Classification is consciously or unconsciously imposed by the person presenting information to the data bank. Not more than six different qualifiers each chosen from an alphabet of ten or fewer letters or words need be specified to enter or retrieve a vector or time series. This estimate can serve as a realistic guideline in the structuring of the index for a properly organized, practical data bank system.

ESTIMATES OF VALUES OF PERFORMANCE PARAMETERS

The rate of arrival of requests for searches of the data bank is most important, and in fact, is a self-dependent parameter. Whenever the value of results of the search, measured in terms of its completeness and freedom from errors, compared with cost and time required is high, the arrival rate for requests will increase. This will in turn degrade the subsequent service of any system operated close to its ultimate capability. For planning purposes, one might arbitrarily select a request rate which is consistent with the economics of storage for the various data sequences in the data bank system. On the assumption there are between 10^4 and 10^6 documents or time series stored in the system when in operation and that such a system might cost \$1000 to \$10,000 per year to operate, it seems reasonable to expect a request rate of 1000 to 10,000 per year or $\approx$30-300/day or $\approx$0.1-1.0/minute.

Priority categories for search could allow for a range of search services and responses ranging from immediate with the system dedicated to a single user to delayed service permitting economization resulting from periods for the accumulation or batching of search arguments. Six appropriate priority categories might be (a) no waiting; (b) 1 minute; (c) 6 minutes; (d) 30 minutes; (e) half day; and (f) daily.

Document retrieval rate might be estimated as 0.1 to 0.5 times the search rate. The number of classes of document display will depend on what output terminal devices are available. These might include the printed page, magnetic tape, paper tape, cathode ray tube photographs, and incremental plotters. Thus, control signals or interfacing with at least six different output devices must be provided.

DESIGN GOALS OR OBJECTIVES FOR A FUNCTIONAL SYSTEM

The central reason for constructing a data bank system is the concentration or channeling of the output from a number of acquisition systems into a single channel that leads to unique storage and complete updating functions. The systems make nearly simultaneous modifications of what is stored and make it feasible to detect duplication of information by independent sources. It then becomes possible to determine if specific information has been previously introduced. These capabilities of a data bank system provide the information needed for generating reports on the current state or past changes in certain resources. The ability to determine which data are not present provides the information directly needed for planning data collection programs. Using the system to provide the amount and location of resources makes it possible for various strategies of resource management to be optimized with respect to certain objectives. In the case of exhaustible or unreplenished resources, the depletion rates of the remnants may be quantitatively monitored. This information can then be used for making policy with regard to the updating of records of licenses or other legal rights to resource utilization.

FUNCTIONS OF COMPUTER DATA BANK SYSTEM

Three broad classes of functions that the system must perform with the information handled by it are: (1) storage, (2) retrieval, and (3) reporting. The first function, storage, has two subdivisions, namely (a) the classification of a series and (b) the writing of its elements into machine addressable locations. The classification aspect of storage facilitates the retrieval and introduces local coherence into the storage system that will permit data with similar space and time coordinates to be obtained with a minimum of reaccessing of the storage medium. The writing aspect of storage is the production of a machine-readable change of state at the addressed locations of memory.

The retrieval of information proceeds in two phases. The first, which is referred to as a search, identifies the entries which satisfy the given search arguments according to some criterion and indicates some of the descriptive properties of each, such as the location, size and the nature of contents. The extraction or display phase of retrieval then copies the set or subset of desired entries to an output unit which may make an electronic record, a physical punched or printed record, or a temporary image such as a cathode ray tube display.

The third function, that of reporting, involves an automatic use of the retrieval function for the purpose of generating reports with statistical or other measures summarizing the data present or indicating what data are missing in a specified region and time interval.

FEATURES REQUIRED IN A PRACTICAL IMPLEMENTATION

The central feature of a data bank system that gives rise to its value is the provision for sharing single copies of data, programs and processing units. The system can remain useful and manageable if it can be insured that only one internal and one or more external backup copies of a specific set of data from any given source can be retained. Also, storage wastage must be minimized by the elimination of unallocated locations through the use of chaining vectors and list structures. The locations of a specific file entry or retrieval point should not be available without a protective password. A provision for enciphering and deciphering the data in sensitive files using pseudo-random number sequences should be available (Carrol 1970). The user should be provided with a facility for creating personal index files to be created for individual user's own applications. Data should be accepted by the system only if it is accompanied by an automatic purge date. All active system files should be backed up by external self-contained files with both table of contents directory and the data records.

OVERALL VIEW OF A DATA BANK SYSTEM

The key feature of a natural resources data bank system is the high degree of structuring derived from the geographic proximity in natural networks and environmental factors such as geologic composition, soil types and rainfall patterns. The total data bank system involves the three components (Figure 2); (1) personnel, (2) programs, and (3) hardware or equipment. The personnel constitute an interface between the acquisition system and the input devices. They also demand that the system stores, searches for, and retrieves data. The programs serve as a vehicle for instructing the hardware to accept instructions from the users. Additional functions that can be performed by the programs include directing further data acquisition, reducing data, and preparing summaries and reports (Figure 3).

It is expected that user's requirements can continue to be met by direct communication of data from the acquisition system and augmented increasingly by the computer-based data bank system. The latter should permit the generation of a central data depository making possible unique and complete updating as well as automation in search and retrieval functions.

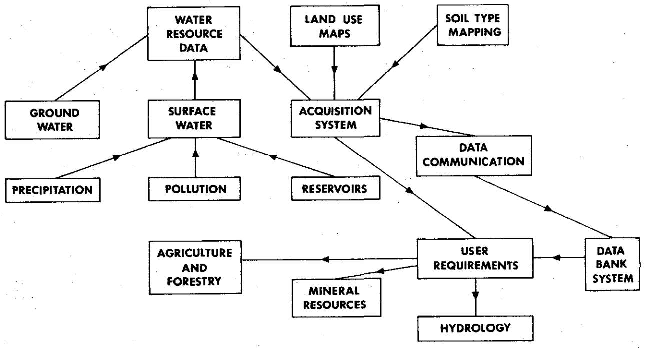


Figure 2

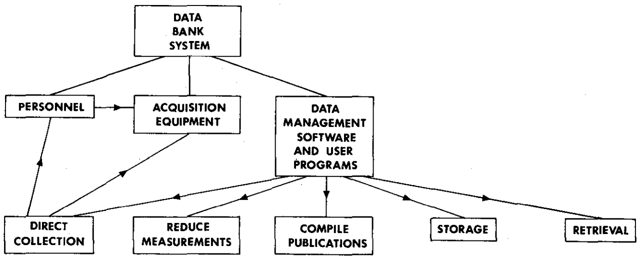


Figure 3

The creation of a data bank system, because of changes in needs and computing systems, must be evolutionary. It is proposed that effective steps are: (1) review of other viable systems, (2) design a system based on data acquisition and unmet needs, (3) simulation of the proposed system design, (4) implementation and use, and finally (5) a redesign or extension repeated recursively, followed by steps 3 through 5.

ENTITIES RECOGNIZED IN THE DATA BANK SYSTEM

One may recognize four distinct classes of natural resources or environmental data, namely: (1) observatory sequences, (2) labeled document-type objects, (3) graph or mathematical tree structures, and (4) area boundaries. Each entity upon undergoing appropriate transformations can be reduced to series to be stored by a computer. The observatory sequences are usually numeric and require little altering other than the affixing of appropriate labels. The sequencing of the elements or values constituting these entities remain, at least within each stored data member, as supplied by the acquisition system. Labeled objects refer to documents with two specified source dimensions. For storage purposes their representation must be decomposed into one-dimensional bit sequences which can be used to reconstruct a display of the source. The graph structures of most frequent interest in natural resources systems are the trees representing drainage networks. These may be decomposed into a binary vector specifying the topology of a particular network and auxiliary vectors corresponding with the types of data associated with each link of the network as shown in Figure 5(a). All such vectors have the same length. Consequently, storing of graphs this way reduces to storing matrices or families of regular sequences. Finally, the fourth distinct entity that arises in natural resources systems are area boundaries. Vectors of equal length may also be used for specifying these by choosing

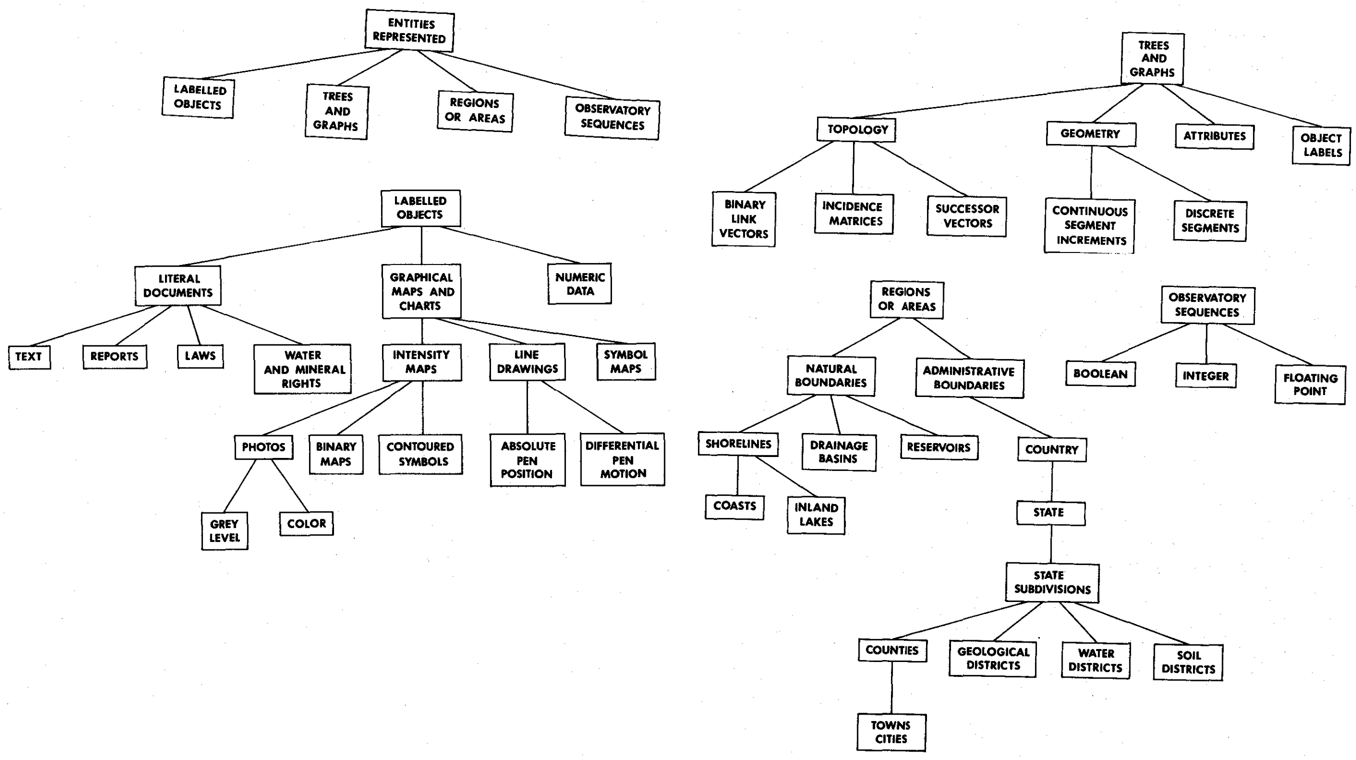
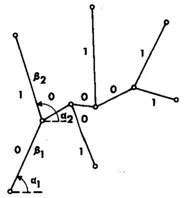


Figure 4



TOPOLOGY-BOOLEAN	0	1	0	0	1	0	1	1	1
DIRECTION-NUMERIC	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9
LENGIHS-NUMERIC	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9

Figure 5a

to approximate a boundary of an area with a sequence of linear segments forming a closed polygon. A pair of vectors with length equal to the number of linear segments can provide the values of coordinate pairs at the terminus of each segment as shown in Figure 5(b). The current word size of computers allows adequate resolution for specifying the segments with all the precision of currently used survey data.

AN ORGANIZATIONAL TREE REDUCING DATA ENTITIES TO SERIES

Data, independent of the form in which it is presented to a computer-based system, must for purposes of computer manipulation be reduced to sequences of bits representing characters or numeric words. The effectiveness with which these data can be retrieved and manipulated depends on how well adapted to the applications is its reduction to the internal form stored within the computer. Economy and rapidity in the manipulation of data depend largely on the data structures chosen for representing the entities of interest.

A practical system should allow the naturally sequenced time series type of data such as observatory records to be stored in their numeric form as directly as possible. A data entity such as time series after being labeled need only have its file entry indexed with respect to geographic location, time, and appropriate documentation.

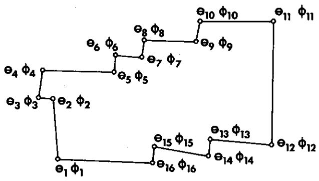
Another type of data entity, referred to here as labeled objects, may be utilized for storing such documentation. These labeled objects may also take on a number of different forms of literal, numeric or graphical information. The common feature of the members of this entity is their internal structure being essentially the sequences of markings and control information required to reconstruct charts or pages of symbols, print, tables, or line drawings. The possibility of the further decompositions of the labeled object type of entity is shown in Figure 4.

Natural resources are usually associated with locations or networks that are concentrated at points or along lines. The points are frequently linked to at least their nearest neighbors by some relationship described by mathematical graph representations which are very often tree structures. An example of a usually low area-density tree is the system of converging channels which make up drainage basins. It is necessary that in the acquisition and storage of the data associated with such systems there be recognized a graph or tree type of entity to provide a consistent and organized way of coding and storing their linked data in series form. It is then possible to formulate algorithms to perform operations defined with respect to the systems they constitute. For example, one might define operations which would correspond with the transport of pollutants from a point in a network to downstream points and identify the latter. Similar tree structures may be needed for storing administrative linkages important in organized data acquisition.

Finally, the fourth entity or data type recognized to be important in geophysical and natural resources information is that of the geographic area which is most economically delineated by describing the path of its boundary. These areas and the boundaries representing them may be natural such as those of watersheds and soil or rock types or artificial such as those representing administrative districts or land use patterns. In many cases the areas overlap or have important memberships in larger areas. Such relationships can be established using appropriate programs designed to operate on series representations of the areas. These representations may be formed out of vectors containing sequenced pairs of numbers corresponding to boundary segments.

CONCLUSIONS

The organization structure identifies a natural resources data bank system with flexibility to accom-



VERTEX COORDINATES θ_1 ϕ_1 θ_2 ϕ_2 . . . θ_n ϕ_n

Figure 5b

modate environmental data within a standardized framework designed to insure low redundancy and high accessibility for regionally distributed data. The central thesis for efficient storage and retrieval is a classification based on the type of format or medium available from the acquisition source and an indexing based on the geographic and time frames of the data. Four types of data entities are proposed, namely: (1) observatory records, (2) labeled objects, (3) tree structures, and (4) geographic areas. Each entity of this classification is decomposed to terminal members which point to the stored sequences or time series-like vectors. Such classification structure is aimed at ideally structured hardware independence to be exploited for the efficient distribution or selective dissemination of geophysical data.

The current acquisition methods and uses of environmental data require a computer based storage system that is organized to facilitate selective retrieval and mapping or data displays. Important features of the system include program controlled purging of obsolescent data and objective program-implemented measures that determine the potential value of particular data members. These two features are needed for the judicious release and allocation of computing resources. The actual implementation of such a system will always be some compromise among cost, access time, integrity of data, and the ease of use. Ease of use refers primarily to data and updating but it is also necessary that requesting a search or retrieval be simple. Search programs serve either to locate available data or indicate data not available. The retrieval function may be used for acquiring statistics on data values, for monitoring purposes, scientific studies, or administrative report generation.

REFERENCES

1 B H BOOM
Some techniques and trade-offs affecting large data base retrieval times
Proceedings ACM National Conference 1969 pp 83-95

2 J M CARROL P M McLELLAND
Fast 'infinite-key' privacy transformation for resource-sharing systems
AFIPS Conference Proceedings Houston Vol 37 1970 November 17-19 1970

3 G DODD
Elements of data management systems
Computing Surveys ACM Vol 1 No 2 1969

4 H T FISCHER
Synagraphic mapping system (SYMAP)
Harvard Laboratory for Computer Graphics

5 R L SHELTON
A photo interpretation and computer graphics for land use and natural resources inventory
Center for Aerial Photographic Studies Cornell University Ithaca New York American Society of Photographmetry Proceedings of 34 Annual Meeting 1968 March 10-15 pp 198-204

6 D H SWANN P B DuMONTELLE R F MAST L H VAN DYKE
ILLIMAP—A computer-based mapping system for Illinois
Circular 451 1970 Illinois State Geological Survey

THE COMMAND TERMINAL—A computerized law enforcement tool

by DAVID M. HUDAK

Automation Technology, Incorporated
Champaign, Illinois

INTRODUCTION

The most crucial problems being faced by local law enforcement agencies today are the collection, analysis, and utilization of police-oriented data. Information is the life blood of any law enforcement agency, and there exists a continuous requirement to collect and produce data which provide information useful to operation, planning, records maintenance, and management decision making.[1]

Police data may occur in many forms; i.e., contact reports, patrol observations, fingerprint files, arrest reports, warrant files, or other identification files. Government agencies at the federal and state level are currently collecting information and making it available to local law enforcement agencies via police teleprocessing techniques (see Figure 1). The NCIC (National Crime Information Center) operated by the Federal Bureau of Investigation in Washington, D.C. is an example of this type of system. It can best be described as a computerized index to documented police information concerning crime and criminals of nationwide interest.[2] Over 25 states now have their own computerized law enforcement systems modeled after the NCIC.

With information pertaining to criminals and criminal activity being maintained at the federal and some state levels, it is to the advantage of police officers to have access to that information whenever possible. If a patrol officer at the scene of an incident or investigation is able to query these data banks at the state and federal levels for the "wanted" status of vehicles or people, he will be able to use this information to make a better judgment of his situation. The timeliness of this inquiry and subsequent response is extremely important, and should be typically less than three to four minutes. Unfortunately, most of the systems currently in use do not accomplish this in an efficient manner, and thus the prime user, the patrol officer, is discouraged from using the system because he usually cannot obtain responses in the time frame that he requires.[1]

THE DISPATCH CENTER

The patrol officer's contact with supporting resources is through the radio or dispatch center. Almost all of his assignments and emergency directives come to him via radio communications and under the control of a radio dispatcher.

Radio dispatchers, monitoring the activities of a widely scattered fleet of vehicles, are the recipients of a vast amount of vital information which has to be sifted, organized, and made available for use in making operational and administrative decisions. The dispatcher's effectiveness depends critically upon the ability of his information handling systems to supply critical data as quickly as possible, with a minimum of wasted effort.

The functions of the dispatch center have changed tremendously in recent years. With the advent of modern communication techniques which can effectively place a two-way radio on every patrol officer, and with

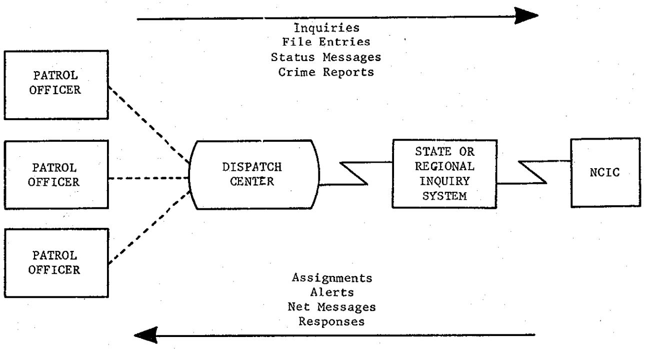


Figure 1—General information flow

the mobile capabilities of law enforcement forces such as patrol cars, motorcycles, scooters, and even helicopters, the dispatch center has become a scene of busy and sometimes frenzied activity, with a corresponding increase in paperwork.[4] Most radio rooms or dispatch centers are not prepared to handle the current volume of activity that exists in law enforcement agencies today. They are often overcrowded, understaffed, and are usually working under an antiquated system that has been "updated" only occasionally over the past years. Although the dispatcher is the field officer's link to other law enforcement resources, responses to their requests for information are frequently delayed minutes, and sometimes hours.

Another important function of the dispatcher accepting complaints or requests for service from citizens is assigning the proper personnel to those incidents from a list of available resources. Recent studies show that 65 percent of all crimes can be solved if the police respond within two minutes of an occurrence of the crime, but if they are delayed up to five minutes, this efficiency drops to 20 percent.[5] In making these assignments, the dispatcher usually makes use of policy and procedure guidelines, experience, and sometimes visual aids such as status boards. It is necessary that he have at his finger tips the status, location, and degree of availability of all of his forces, in order to assign the correct resource to each task.[6]

RADIO TICKETS

The flow of data through a dispatch center is usually controlled by paper forms called radio tickets (or dispatch tickets, complaint tickets, etc.). This is a data collection method that has been used in police departments throughout the country for a number of years. As much as they may vary from department to department, radio tickets are generally used to record some basic facts about each assignment, event, or incident that requires the use of the police departments' primary resource, manpower.[7] Some of the most common elements of the radio ticket include: time complaint was received, type of incident, location of incident, unit number assigned, time dispatched, time of arrival on scene, a document control number, time assignment was concluded, etc. These tickets may be completed entirely by the dispatcher as the incident progresses, or may be later filled in by the patrol officer when he returns to the station. In larger departments the radio tickets are initiated by complaint clerks who take the complaints from the public and pass them on to dispatchers, who assign the mobile units and complete the radio tickets.

Current manual radio tickets provide a very important collection device in the data management procedures of a law enforcement agency. However, there are some serious drawbacks to their use. For instance, if the data are going to be input to a law enforcement management information system, then the information on the radio ticket must be recorded twice, once by the dispatcher as it occurs, and again by a clerk who codes the information into machine-sensible form. In some cases the physical environment of the dispatch room requires that some mechanical method be used to transport the radio ticket from one position to another, thereby increasing the time from receipt of complaint to the time the unit is dispatched.

Should a patrol officer wish to make an inquiry on a wanted person or car, the request for information must be logged on a radio ticket, and the dispatcher must relay the request via a terminal to the state or regional data base inquiry system. The dispatcher must input the inquiry in a very rigorous format, because errors will cause it to be rejected and necessitate its reentry. Not only is this activity frustrating to the operator of the terminal, but in emergency situations it could endanger the patrol officer who has asked for this information NOW and is unable to get it.

It is readily apparent that there are many stumbling blocks between the information system and its user, the officer on patrol. The system often appears unresponsive to his needs, so he tends to use it only when absolutely necessary, and not as part of a normal operating sequence. To cope with these problems, the attack must be launched at the nucleus of the difficulty, the radio room itself, its operating personnel and procedures, and the operator-information system interface. The radio dispatcher must be provided with an integrated mechanism for handling his normal dispatching and data collection duties, and also for interfacing with the automated information system in an efficient manner. A system has been constructed which will solve many of these problems. This system is called the COMMAND TERMINAL.

FUNCTIONS OF THE COMMAND TERMINAL SYSTEM

The basic COMMAND TERMINAL System is composed of a minicomputer, a disk, a visual display device (CRT), a standard keyboard unit, a special function keyboard unit, a standard teletypewriter, and a set of operational computer programs. While the system is welded together logically into a total operating package, the dispatcher is physically confronted with only the visual display device, the two keyboards, and infrequently, the teletypewriter (see Figure 2).

In many cases the visual display device can be integrated into the radio console itself, so that it is immediately in front of the dispatcher when he is seated in his normal operating position. The two keyboards are placed on the console desk top in front of the dispatcher. The teletypewriter, which is used primarily for hard copy output, may be located in various places, depending upon the department's normal operational procedures. All other equipment may be located in a separate room with the communications equipment, or in any other suitable place. The system is designed to replace, on a plug-for-plug basis, the present real-time information system terminal. Since the COMMAND TERMINAL can emulate the previous type of terminal, no modification to the real-time system's programs is necessary.

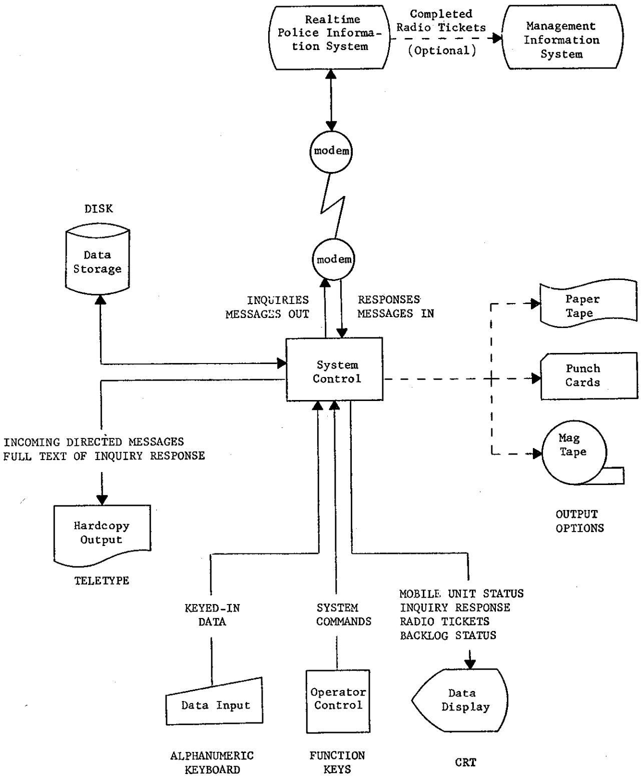
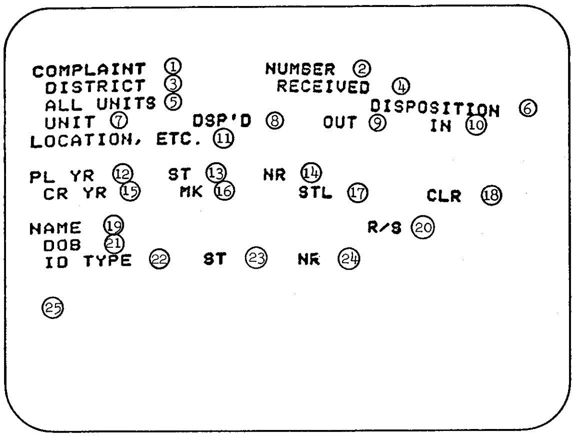


Figure 2—Information flow for COMMAND TERMINAL SYSTEM

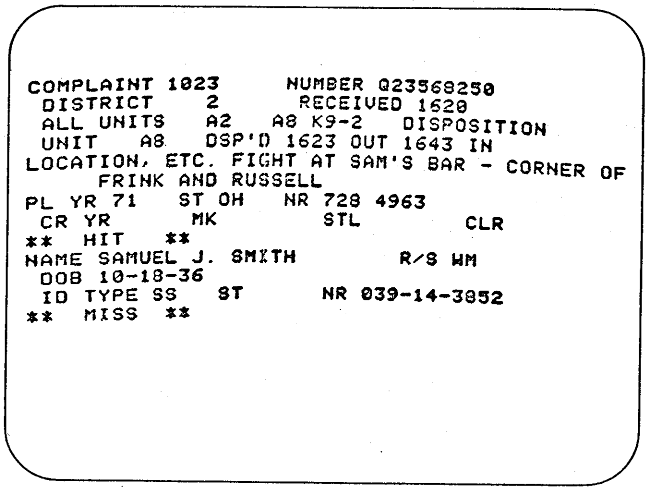
With the COMMAND TERMINAL, radio tickets are no longer pieces of paper to be completed in handwriting, time stamped, filed in slots, and ultimately keypunched to permit further analysis by computer. Instead, the radio ticket is an electronic form appearing before the dispatcher on a display device. The dispatcher fills in some of the blanks by utilizing a keyboard, while the system assists him by filling in others automatically (see Figure 3). In some cases, when data fields have been filled in by the dispatcher, an auto-



```
COMPLAINT ①          NUMBER ②
 DISTRICT ③           RECEIVED ④
 ALL UNITS ⑤                    DISPOSITION ⑥
 UNIT ⑦       DSP'D ⑧    OUT ⑨    IN ⑩
LOCATION, ETC. ⑪

PL YR ⑫    ST ⑬    NR ⑭
 CR YR ⑮    MK ⑯        STL ⑰        CLR ⑱

NAME ⑲                          R/S ⑳
 DOB ㉑
 ID TYPE ㉒    ST ㉓    NR ㉔

㉕
```

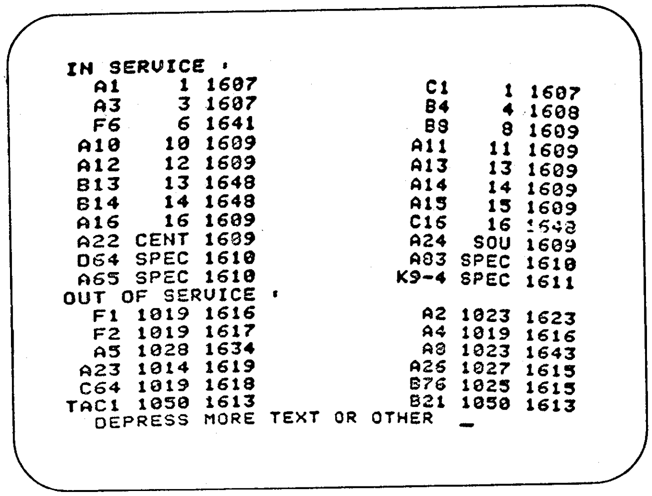
1	Type of complaint	14	License Plate Number
2	Document Control Number	15	Model Year of Car
3	Applicable Patrol District	16	Make of Car
4	Time Complaint Received	17	Style of Car
5	Units Responding to this Incident	18	Color of Car
6	Disposition Code	19	Name of Person
7	Mobile Unit Identifier	20	Race/Sex
8	Dispatched Time	21	Date of Birth
9	Out-of-service Time	22	Type of ID Number Following
10	In-service Time	23	State, if Driver's License Number
11	Location of Incident	24	ID Number
12	License Plate Year	25	Additional Comments
13	License Plate State		

Figure 3a—Blank radio ticket



```
COMPLAINT 1023      NUMBER Q23568250
 DISTRICT    2       RECEIVED 1620
 ALL UNITS   A2   A8 K9-2   DISPOSITION
 UNIT   A8   DSP'D 1623 OUT 1643 IN
LOCATION, ETC. FIGHT AT SAM'S BAR - CORNER OF
      FRINK AND RUSSELL
PL YR 71   ST OH   NR 728 4963
 CR YR       MK          STL          CLR
**  HIT   **
NAME SAMUEL J. SMITH            R/S WM
 DOB 10-18-36
 ID TYPE SS   ST        NR 039-14-3852
**  MISS  **
```

Figure 3b—Radio ticket in progress



```
IN SERVICE ,
  A1     1 1607                     C1     1 1607
  A3     3 1607                     B4     4 1608
  F6     6 1641                     B8     8 1609
 A10    10 1609                    A11    11 1609
 A12    12 1609                    A13    13 1609
 B13    13 1648                    A14    14 1609
 B14    14 1648                    A15    15 1609
 A16    16 1609                    C16    16 1648
 A22  CENT 1609                    A24   SOU 1609
 D64  SPEC 1610                    A83  SPEC 1610
 A65  SPEC 1610                   K9-4  SPEC 1611
OUT OF SERVICE ,
  F1  1019 1616                     A2  1023 1623
  F2  1019 1617                     A4  1019 1616
  A5  1028 1634                     A8  1023 1643
 A23  1014 1619                    A26  1027 1615
 C64  1019 1618                    B76  1025 1615
TAC1  1050 1613                    B21  1050 1613
  DEPRESS MORE TEXT OR OTHER  _
```

Figure 4—Mobile unit status

matic edit is performed by the system to check the validity of the data.

Tickets may be temporarily filed away in the system's computer memory at any time to permit the dispatcher to process or initiate other tickets. These stored-away tickets may be recalled at will by the dispatcher until such time as the particular assignment has been concluded and the mobile unit is back in service.

A mobile unit status display is maintained on an up-to-the-minute basis by the system as a by-product of processing the radio tickets. Thus, the dispatcher can at any time press one function key and review the in- or out-of-service status of all cars under his command, along with some additional facts about various units (see Figure 4).

As the dispatcher records the particular beat or patrol area identifier for an incoming complaint, the system will automatically recommend a specific mobile unit to handle the incident. The recommendation is based on a priority structure established initially by the department and, naturally, also considers the present availability status of all units. On more sophisticated models of the system, it will be possible to automatically translate a street address into the appropriate beat number, thereby saving the dispatcher this task.

The dispatcher also has the option of placing a new incident on backlog status, instead of assigning it immediately to a unit. If he does, the system automatically assigns the ticket a priority based on the type of incident, and backlogs it for future reference. Subsequently, it will be brought to the attention of the dispatcher as a logical unit becomes available. A status report on the backlog can be viewed at any time by the dispatcher (see Figure 5).

As name and vehicle checks come in from the mobile unit handling a specific incident, the dispatcher need only fill in the necessary fields on that unit's radio ticket. The system then automatically formats this data into the proper inquiry language, and transmits the resultant inquiry to the real-time system.

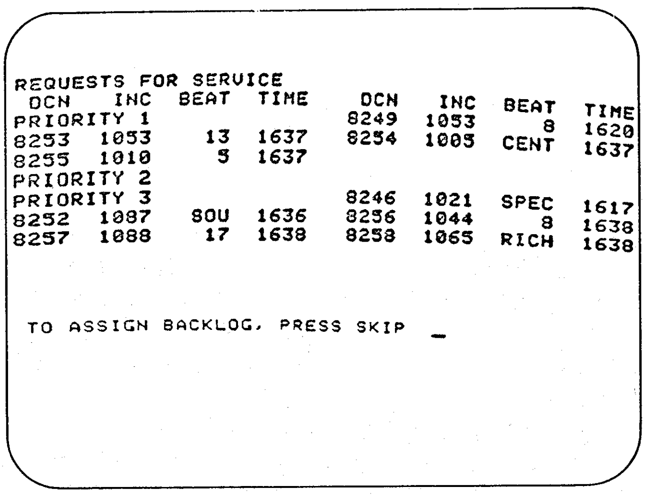
As a safety feature for the man on the street, the system contains a "watchdog timer" which notifies the dispatcher if no contact has been made with a particular unit within some predetermined amount of time. The amount of time permitted before an alert message is displayed can vary, depending upon the type of incident being handled.

Complete radio tickets, being recorded originally in machine-sensible form, can be sent directly to the management information system computer if it is capable of accepting data this way, or it can be recorded on any type of output medium desired, such as paper tape, punched cards, or magnetic tape.

Directed or point-to-point messages coming to the agency through the real-time-information/message-switching system are automatically routed for output to the teletype by the COMMAND TERMINAL's small computer. This precludes interference with the dispatcher's normal operation. Outgoing directed messages can be sent from the dispatcher's visual display unit, or the teletype. In either event, the system will handle all necessary formatting and the insertion of fixed message header elements.

SYSTEM CONFIGURATION

As mentioned before, the basic system consists of a minicomputer, a disk, a CRT with a standard keyboard,



```
REQUESTS FOR SERVICE
 DCN    INC  BEAT  TIME      DCN    INC   BEAT   TIME
PRIORITY 1                   8249   1053     8   1620
8253   1053    13  1637      8254   1005  CENT   1637
8255   1010     5  1637
PRIORITY 2
PRIORITY 3                   8246   1021  SPEC   1617
8252   1087   SOU  1636      8256   1044     8   1638
8257   1088    17  1638      8258   1065  RICH   1638



 TO ASSIGN BACKLOG, PRESS SKIP  _
```

Figure 5—Backlog status

and a function keyboard. One of the design goals in the development of the COMMAND TERMINAL was to achieve a satisfactory price/performance relationship with a sound functional system. The Honeywell H316 computer was selected as the system CPU. The Engineered Data Peripherals 3032 disk was interfaced in-house, and the basic sector size set at 128 16-bit words. This sector size was defined so as to contain a single radio ticket with an average number of entries, and to minimize the number of sector transfers when executing program overlays. The CRT used is a TEC Model 400 with a complete set of edit functions and standard alphanumeric keyboard. The special function keyboard was built in house and merely converts the dispatcher's finger stroke into an 8-bit character. All of the system functions are implemented by programming.

All of the computer programs were coded in DAP, the machine assembly language of the Honeywell computer. A disk-based monitor was designed to control the disk-resident overlay processors and data storage. Almost all of the data being handled are maintained on the disk. This includes completed radio tickets, radio tickets in progress, unit status tables, patrol area status tables, input and output message queues, and other user-specified information that needs to be maintained in real-time.

Due to the variety of data base structures and the individuality of each law enforcement agency, a major design goal was modularity of function programs. It is now a minimal effort to design and implement a tailor-made system to any agency's specifications. Parameters such as radio ticket format, patrol area labels, response patterns, and even incident priorities can be specified during an installation by a short system generation procedure.

EXPANDABILITY

The basic system is easily expandable both in core and disk size. For larger installations that require more than one dispatcher station, additional CRTs and keyboards may be added to the system (see Figure 6). However, each additional dispatcher station approximately doubles the processing requirements of the system, and therefore usually requires additional core and disk space in order to guarantee acceptable interaction of the dispatcher with the system. "Acceptable Interaction" is defined as the system's response to data input or action by the dispatcher, and has a maximum time lapse of two seconds. The COMMAND TERMINAL interface to a police real-time information system is usually via a modem and requires only specific formatting of already developed character streams, and is, therefore, relatively easy to implement.

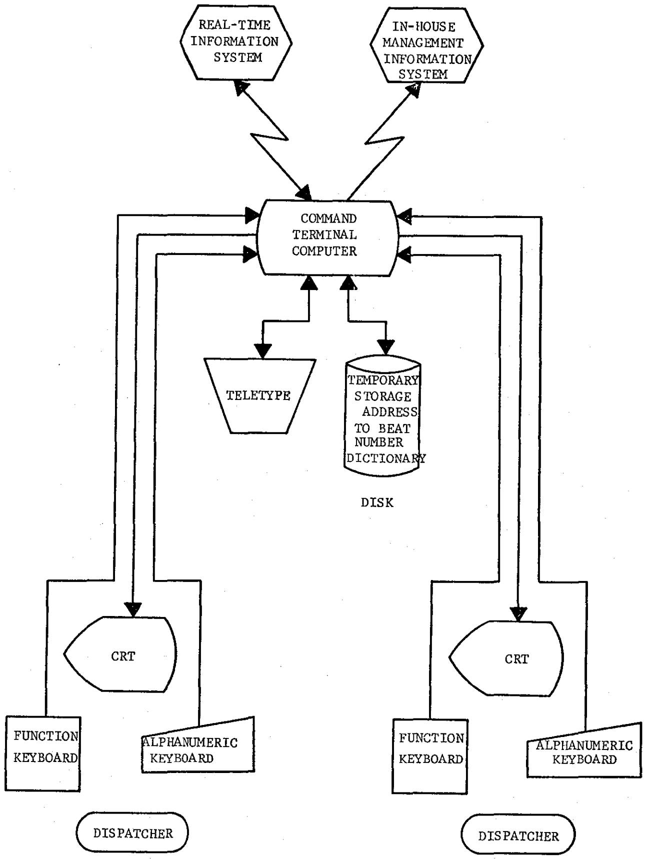

REAL-TIME INFORMATION SYSTEM
IN-HOUSE MANAGEMENT INFORMATION SYSTEM
COMMAND TERMINAL COMPUTER
TELETYPE
TEMPORARY STORAGE ADDRESS TO BEAT NUMBER DICTIONARY
DISK
CRT
CRT
FUNCTION KEYBOARD
ALPHANUMERIC KEYBOARD
FUNCTION KEYBOARD
ALPHANUMERIC KEYBOARD
DISPATCHER
DISPATCHER

Figure 6—Medium-sized city configuration

SUMMARY

We have described a system that has been designed to facilitate the flow of police information through the dispatch station of a typical law enforcement agency. Particular attention has been given to system modularity, flexibility, and ease of modification for varied installations. Use of a general purpose computer as the system controller guarantees ease of expandability when future "add-ons," such as mobile teleprinters and automatic vehicle monitoring devices, become available. These two items would serve to automatically close the loop for an integrated command-and-control system.[9]

The COMMAND TERMINAL has proved itself a capable tool in a real-time computer-oriented dispatch environment. It is valuable because any tool that collects and organizes data in a complicated and changing fact situation is a significant aid to judgment.[10] And that is the goal—to provide a tool for our law enforcement officers.

REFERENCES

1 P M WHISENAND T T TAMARU
Automated police information systems
John Wiley & Sons Inc New York 1970

2 Federal Bureau of Investigation
NCIC operating manual
Revision G September 1969

3 P J PITCHESS
New computer based dispatching system cuts response time
The Police Chief Volume 38 No 2 pp 8 58-59

4 E G COLUMBUS
Management by systems
The Police Chief Volume 37 No 7 pp 14-16 July 1970

5 C M KELLEY
News article
Datamation p 16 December 1970

6 R J RIEDER
Command and control for law enforcement
The Police Chief Volume 37 No 7 pp 24-29 July 1970

7 P M WHISENAND T T TAMARU
Automated police information systems
Chapter 6

8 D M HUDAK
(Internal Communication)
Automation Technology Incorporated 1970

9 S A YETSKY J D HODGES JR
Field communications for command and control
The Police Chief Volume 37 No 7 pp 34-42 July 1970

10 P M WHISENAND T T TAMARU
Automated police information systems
pp 198-199

Experience gained in the development and use of TSS

by RICHARD E. SCHWEMM

IBM Corporation
White Plains, New York

INTRODUCTION

Six and a half years have elapsed since W. T. Comfort described TSS/360 to the 1965 Fall Joint Computer Conference.[1] Since that time, much has been learned by IBM and its customers about time-sharing, about TSS, and about large-scale, interactive systems in general. Scores of people have worked with the system; dozens of articles have been published; it would clearly be impossible to put in one paper a comprehensive answer to the question—what has been learned developing TSS? Yet, with the availability of Release 8.1, the major development work on the system has been completed, and this is an appropriate time to take stock of where we have been, where we are, and where we might go from here.

One summary paragraph of that 1965 paper says the following: "This report attempts to give an overall picture of the System/360 Model 67 Time-Sharing System, its system design, and major hardware and control program characteristics. The unique combination of hardware and software objectives makes a very complex problem, for which a simple and efficient solution is desired—a difficult task at best." Indeed, bringing TSS/360 to the marketplace and making it productive in customer installations has been a very difficult task, both technically and financially.

No attempt has been made in this paper to give detailed technical descriptions of programming systems problems and solutions. Instead problems have been described in a general way, their impact on the system discussed, and the strategy for solution outlined along with the results obtained. Four major areas will be discussed:

- Lessons Learned about System Structure
- Lessons Learned about System Performance Analysis
- Lessons Learned about Software Development Tools
- Lessons Learned about Management of Software Development

SYSTEM STRUCTURE

Version 1 of TSS/360 was released in October, 1967. Use for experimental, developmental, and instructional purposes was advised. By the fall of 1968, A. S. Lett and W. L. Konigsford were able to report[2] on a version which was more stable, performed better, and contained more function. Based on this progress made during the first year in the field, plus progress anticipated from items designed and then in development, a very positive view of the system was presented.

Several important lessons had emerged from the TSS experience by this time. It seems clear that the proper way to construct a system of this type is to build a small, hard-core base first, and add function later. As Lett and Konigsford put it—"The initial emphasis was on building a stable system, followed by extensive measurement—and—analysis efforts to identify potential system modifications."

The same paper reported that the design of the TSS Resident Supervisor had proved sound and remained essentially as described in 1965, a statement as valid today as it was in 1968. On the other hand, major portions of the original design had to be abandoned for the simple reason that they were unacceptable to the Users. One example of this was the replacement of the original rigid set of commands with Command System II.[3] Another was the replacement of the system's hard-coded scheduling algorithm with an extremely flexible Table Driven Scheduler.[4] The important thing here is not that the original design was unacceptable, but that a communication channel was open from the Users to TSS Development, and that the latter was responsive to the former's requirements.*

* In both cases cited as examples, Users made strong technical presentations. T. A. Dolotta of Princeton University and A. Irvine of System Development Corporation made a "Proposal for Time-Sharing Command Structure" at SHARE in August, 1966. The basic thinking behind table driven scheduling came from F. G. Livermore of General Motors Research.

The table below displays the functional buildup of TSS/360 by release:

TABLE I—Summary of Major Functions Added to TSS/360

Date	*Release*	*Major Functions Added*
2/1/68	1.1	(Incremental reliability improvements only)
4/15/68	1.2	(Incremental performance improvements only)
7/1/68	2.0	• *Support for Extended Address Translation (32 bit)* On Model 67 equipped with the special feature, users can address up to 4096 virtual memory segments. • *Multiple Sequential Access Method (MSAM)* This is a special access method for card equipment and printers which allows a single task (BULKIO) to process all spooling operations.
10/21/68	3.0	• *Command System II* This system provides a consistent command syntax, an extensive default and profile facility, a general editing capability, and allows users to define their own commands and call programs directly. • *Time Sharing Support System* (*TS*[3]) A Virtual Support Subsystem executes in parallel with other TSS operations, allowing system programmers to access and modify the privileged, shared Virtual Memory; a Resident Support Subsystem provides the same facility for the resident supervisor in a non time-shared mode.
1/20/69	4.0	• *Table Driven Scheduler* This technique provides installation control and dynamic regulation of scheduling parameters, for complete flexibility in obtaining optimum scheduling balance within and among tasks.
7/8/69	5.0	• *Support For Duplex Configurations* This was the first system release which was tested on and suitable for a multiprocessing configuration. • *Virtual Access Method II* This rewrite of VAM provides more efficient handling of physical devices, ability to create duplicate data sets, and an overall improvement to data set integrity.
10/15/69	5.1	(Incremental human factors improvements only)
3/31/70	6.0	• *Resident Terminal Access Method (RTAM)* Removing TAM from shared virtual memory reduces paging which significantly improves conversational task performance. • *Multiple Terminals per Task (MTT)* A large number of users can share the same task and the concomitant overhead. The effect is the addition of a generalized way to add subsystems.
5/15/70	6.1	(Maintenance only)
6/30/70	7.0	• *PL/I Compiler* A version of the OS/360 F-level compiler is modified to operate within TSS. • *Remote Job Entry* Non-conversational job streams can be introduced into TSS from a remotely located work station; spooling to/from the work station is also provided.
12/17/70	8.0	(Incremental improvements only)
10/1/71	8.1	• *Dynamic Catalog* The user portion of the master catalog is made a working catalog for the duration of the user's task, eliminating contention for the master catalog. • *Page Table Paging* Demand paging of page tables allows effective use of very large virtual memories.

This functional buildup was accompanied by a steady improvement in performance and reliability. The performance picture is discussed in detail in a subsequent section. To date, a total of 2442 program errors detected by Users have been corrected.

Figure 1 displays the system size as a function of release. It is worthy of note that only the addition of the PL/I Compiler caused a significant increase in system size.

SYSTEM PERFORMANCE ANALYSIS

The economic viability of any system is determined by a complex equation which combines price, performance, and the value of function.[5] Since a given set of hardware has a fixed price, and a given set of software functions appear to a User to have a constant value, performance is the critical factor in determining success or failure. A simple example will illustrate this:

> *Let us assume that an interactive system operates on a configuration that costs $1,000,000 per year. Let us further assume that users feel that each professional employee using this system will double his productive output and that an average employee of this type costs $30,000. Then the value of the system is a simple function of how many users it can support—if it supports 33 or fewer, it will be a loser, 34 or more a winner.*

During the development of TSS/360, a comprehensive scheme for treatment of system performance evolved. The elements of this scheme are:

- Establishment of Performance Objectives
- Creation of External Performance Measurement Tools
- Creation of Internal Recording Tools and Appropriate Data Reduction Facilities

Performance objectives

How well will the system perform the functions it provides? This simple question receives a complex answer. TSS/360 is designed for conversational use, batch use, and a mixture of the two. A performance objective was established for each type of use. The conversational objective is the most difficult to describe and of greatest interest here, so our discussion will be restricted to it.

Basically, conversational performance is defined as the maximum number of tasks which the system will support with acceptable response time. This very general definition is made more specific by creating a benchmark terminal session and dividing the interactions created by the session into three classes—trivial response, non-trivial response, and data-dependent response. Acceptable response to trivial commands (such as text entry) is defined as four seconds. Acceptable response to non-trivial commands (such as data set creation) is defined as 7.5 seconds. Data-dependent commands (such as compilation) have no specific acceptability criteria. With this definition of conversational performance, we have a constant, calibrated yardstick that answers the performance question.

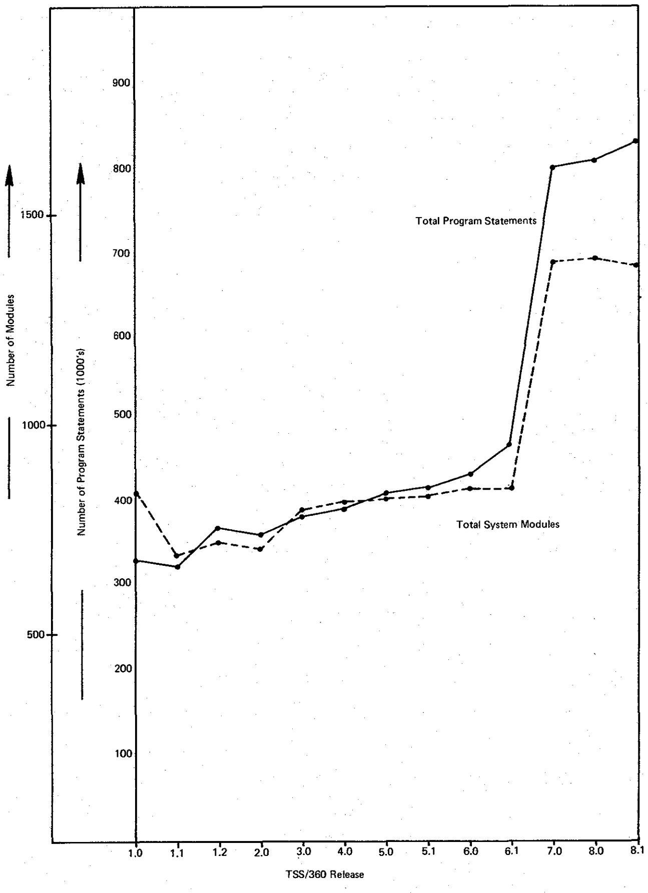


Figure 1—Size of TSS/360 as a function of release

Here we should point out that no User of TSS accepts our benchmark terminal session as typical of his conversational work load. Most have specified their own benchmarks. However, there is general agreement that the definition of performance is adequate. We have a yardstick; the users have metersticks; we are in fact measuring the same thing.

The initial conversational performance objective established for TSS/360 was to support 40 tasks running the benchmark terminal session on a configuration composed of 1 processing unit, 512K bytes of memory, 1 paging drum, and disk modules on 1 channel. Subsequently, objectives for larger configurations were established.

External performance measurement tools

TSS operates in an extremely dynamic environment, and the load imposed upon it by live users is characterized by peaks and valleys of activity and demands for services. Yet, for performance objectives to be useful, they must be measurable, and the measurements must be repeatable. To achieve this, a measurement driver was created to simulate the actual environment under controlled and reproducible conditions.

Shown schematically in Figure 2, the driver has the following characteristics:

- It interfaces with the system in the same manner as actual terminals and, to the system, appears indistinguishable from them.
- It is capable of bringing the system up to a stable load level and keeping it there during the measurement period in order to eliminate starting and stopping transients from a measurement.
- It is capable of recognizing the expected system response for each transmission and contains sufficient logic to take appropriate action for any response received.

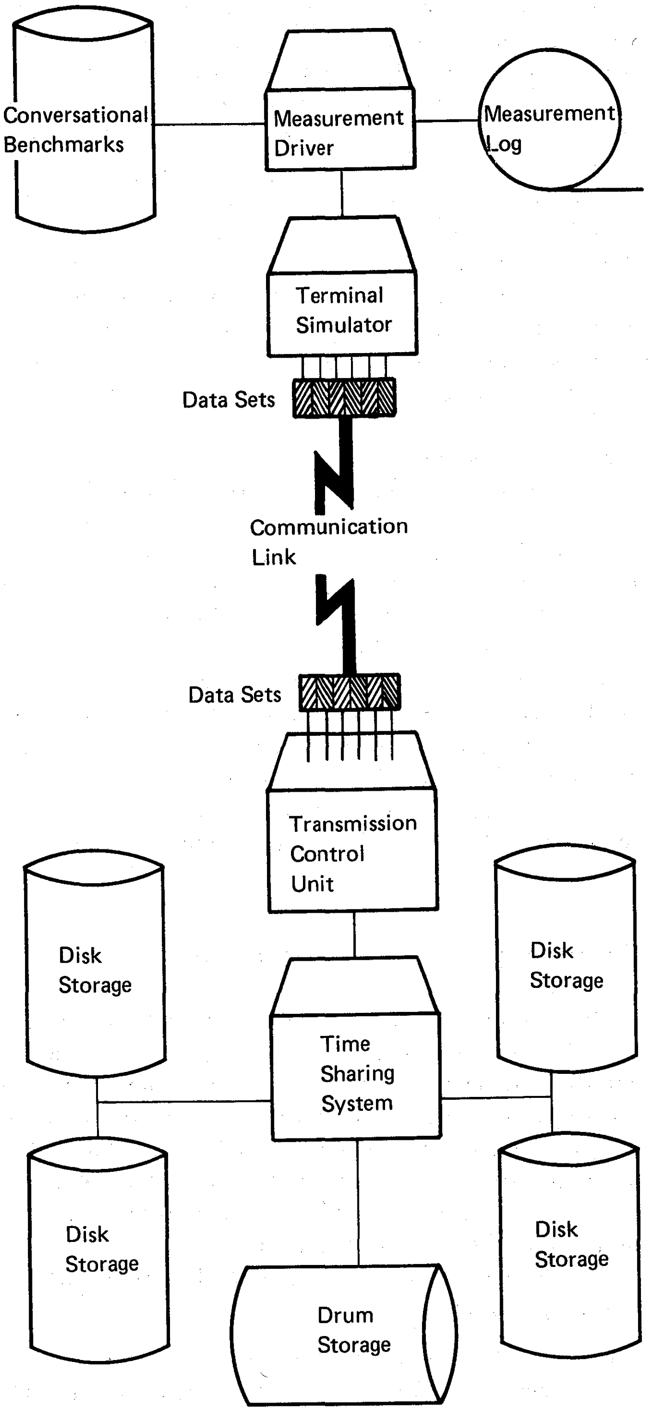


Figure 2—Schematic view of TSS/360 measurement driver

- It is capable of recovery from at least the "ordinary" kind of unexpected response, such as might result from transmission errors.
- It accepts as a parameter, a "human keying rate" for each "terminal" to adjust for the difference between machine-driven transmission rates and real keying rates.
- It records and time-stamps all transmissions in both directions on all lines, together with the control parameters and other such data, to allow data reduction and analysis at a later time.
- It is capable of terminating a run, based upon cut-off criteria specified for that run, to avoid wasteful use of machine time.
- It is data directed in its operation, so that not only the transaction to be transmitted but also the control of the individual delays between each interaction can be specified in the conversational benchmarks, in accordance with the measurement rules.

Conversational performance is measured by executing a series of driver runs varying the number of tasks for each run. A curve is then constructed which represents response time as a function of the number of tasks. Such a curve for TSS Release 6.0 is shown in Figure 3.

The measurement driver developed by IBM runs on a System/360 Model 40. Several Users of TSS have used this system to evaluate system performance on their own benchmarks. A second measurement driver has been developed by Carnegie-Mellon University. This driver has the advantage of running on the System/360 Model 67 with the version of TSS under study. Although compensation must be made for the system resources devoted to the driver function, the Carnegie-Mellon Simulator (SLIN) is compatible in script and timing characteristics with IBM's, and the output produced is comparable.

Internal recording tools

Answering the question "how well is the system performing?" from an external viewpoint provides little insight into the question "how can performance be improved?" For this task, internal recording tools must be used to obtain knowledge of the internal operation of the programming system and of how that system utilizes its hardware facilities. The basic measurement and recording tools used by TSS Development are **S**ystems **P**erformance **A**ctivity **R**ecorder (SPAR),[6] Systems **I**nternal **P**erformance **E**valuation (SIPE),[7] and **I**nstruction **T**race **M**onitor (ITM).

The System Performance Activity Recorder (SPAR) is a one-of-a-kind hardware recording system. It is hardwired to the system being monitored and does not cause any degradation in performance.* SPAR can

* Similar capability is provided by IBM's System Measurement Instrument Unit (SMI).

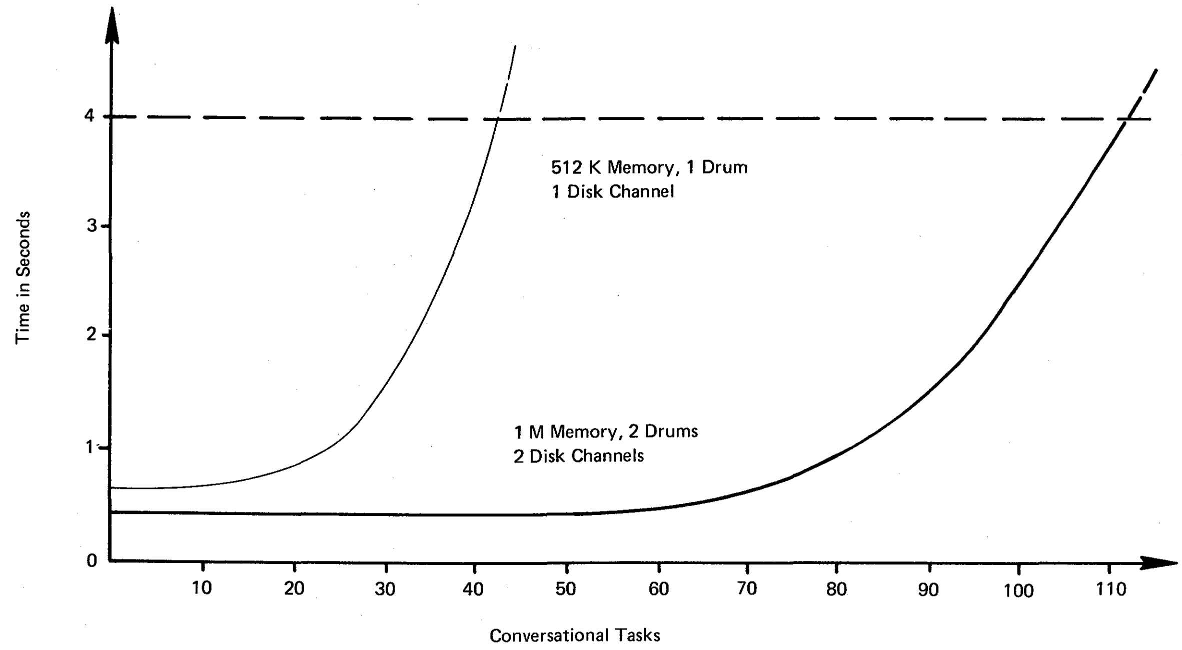


Figure 3—TSS/360 release 6.0 response to trivial commands as a function of number of tasks

provide accurate measurements of system facility utilization. The facilities monitored for TSS/360 include the CPU, processor storage, I/O channels, and direct access devices. SPAR is used to provide the following types of information:

- Time utilization of the system hardware facilities
- Counts and time relationship of indentifiable events such as:
  - Time-slice ends
  - Pages transferred between devices
  - Entries to a module

System Internal Performance Evaluation (SIPE) is a software recording system that produces a degradation of approximately five percent in system performance. Hooks in the resident supervisor cause information to be collected and written on tape. The internal actions of the supervisor are then available for later data reduction. SIPE is used to provide the following types of information:

- Time utilization of system hardware facilities
- Space utilization of processor storage, drum, disk storage
- Counts of system events
- Time relationship of important internal events in the supervisor

The Instruction Trace Monitor (ITM) is a hardware and software combination that causes every instruction executed to be written in sequence on tape. The instruction sequence in resident supervisor or virtual memory is thus available for later data reduction. It should be noted that ITM causes a relatively large degradation in system performance by increasing system running time. ITM is used to provide the following types of information:

- Time sequence of virtual memory modules executed and referenced
- Time spent in each module
- SVC's issued from each module

With these tools, and with an appropriate set of reduction programs, virtually any question dealing with the internal system performance can be answered. Access to information of this type, allows an intelligent, aggressive program of performance improvement to be carried out. The results of such a program are shown in Figure 4.

One final point should be made about internal recording tools—they have been as valuable to Users as they were to the development organization. As mentioned earlier, Users have their unique workloads, and TSS's completely flexible scheduling technique is most effectively used with detailed knowledge of how

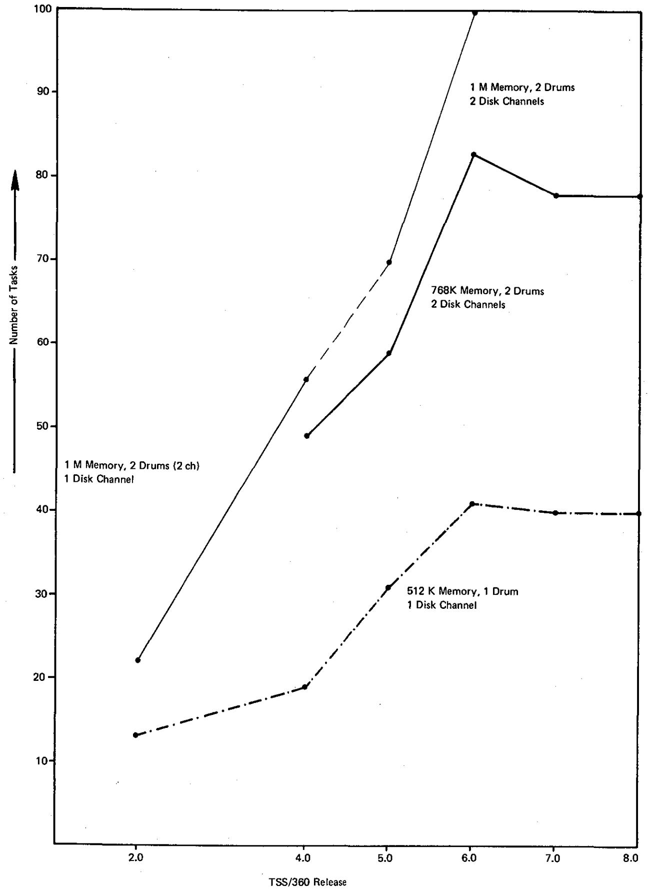


Figure 4—Number of tasks within response time objectives

the system is operating on that individual workload. SIPE is in use at most TSS installations, and several Users have developed their own measurement tools. Of particular interest in this category are DEMON—a real time software measurement monitor developed at Bell Telephone Laboratories, Naperville, Illinois and XDEMON—an expanded version developed jointly by Carnegie-Mellon University and IBM's Watson Research Center.

XDEMON is a time-driven measurement tool providing statistics on the usage of resources in TSS/360. XDEMON works as a shared facility under the system maintaining a set of public tables, whose information is updated automatically and made available to the users, who can display selected information concerning TSS usage accumulated since STARTUP time and for the period since the last update. There is a privileged user, called the 'MONITOR' who has the capability of issuing some selected commands, including the updating of the shared tables, the recording of statistics for later reduction, and the monitoring of tasks that are imposing an excessive load upon the system, with the possibility of changing priorities and scheduling parameters for such tasks.

SOFTWARE DEVELOPMENT TOOLS

"Give us the tools, and we will finish the job." Looking back over six years much can be said about the tools necessary to develop a system such as TSS, and that is the subject matter of this section.

The types of tools utilized can be broken into three general categories:

- Machinery
- Language Translator with appropriate utilities
- Debugging Aids

Little that is unique in the first two categories was done by TSS Development. As in the case of many total systems projects (hardware and software combinations) software development began well in advance of the availability of the System/360 Model 67 hardware. To overcome this situation, a Model 67 Simulator was created to operate on the standard System/360. The TSS system itself is written in Macro Assembler. Initial assemblies were accomplished on a version of the Basic Operating System/360 (BOS/360) Assembler modified to expand TSS macros. Additionally, a utility function was created which produced TSS object module format from the output of the BOS/360 Linkage Editor.

Debugging aids

The initial set of TSS debugging aids was provided by a system called Support for **T**est, **R**elease and **T**ime-Shared **O**perations (STRATO). STRATO was a major modification of BOS/360. It provided three major functions:

- An Interactive Console Command Language
- A Model 67 Simulator
- A Test Case Driver

The STRATO Command Language provided facilities to system programmers to load and unload programs, to start, stop, and dynamically interrupt programs, to display, alter and dump memory. The Test Case Driver would supply inputs to a version of TSS running under the Simulator and record outputs.

Early versions of the TSS Resident Supervisor were hand-built as BOS/360 Jobs and then run and tested under STRATO. Once the Supervisor was cycling, critical Virtual Memory Functions were integrated. Much of this work was completed on Model 40 systems before the first Model 67 was installed.

The second phase in the development of debugging aids consisted of removing the dependency upon STRATO. With the availability of the Model 67, the Simulator function was removed, and STRATO was moved to the Model 67. Integration and Test of the TSS Assembler and Dynamic Loader was the next milestone; the dependency upon the BOS Assembler was removed. STRATO itself was released with TSS, since it remained the only system programmer tool for manipulating the resident supervisor and shared virtual memory.

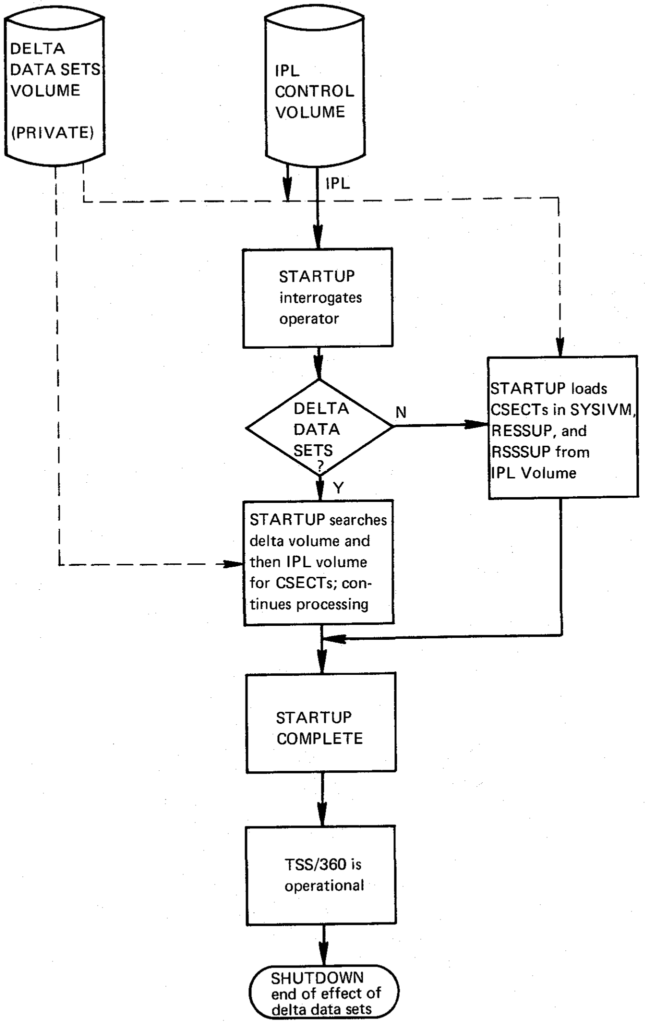

DELTA DATA SETS VOLUME (PRIVATE)
IPL CONTROL VOLUME
IPL
STARTUP interrogates operator
DELTA DATA SETS ?
N
Y
STARTUP loads CSECTs in SYSIVM, RESSUP, and RSSSUP from IPL Volume
STARTUP searches delta volume and then IPL volume for CSECTs; continues processing
STARTUP COMPLETE
TSS/360 is operational
SHUTDOWN end of effect of delta data sets

Figure 5—TSS/360 dynamic modification procedure

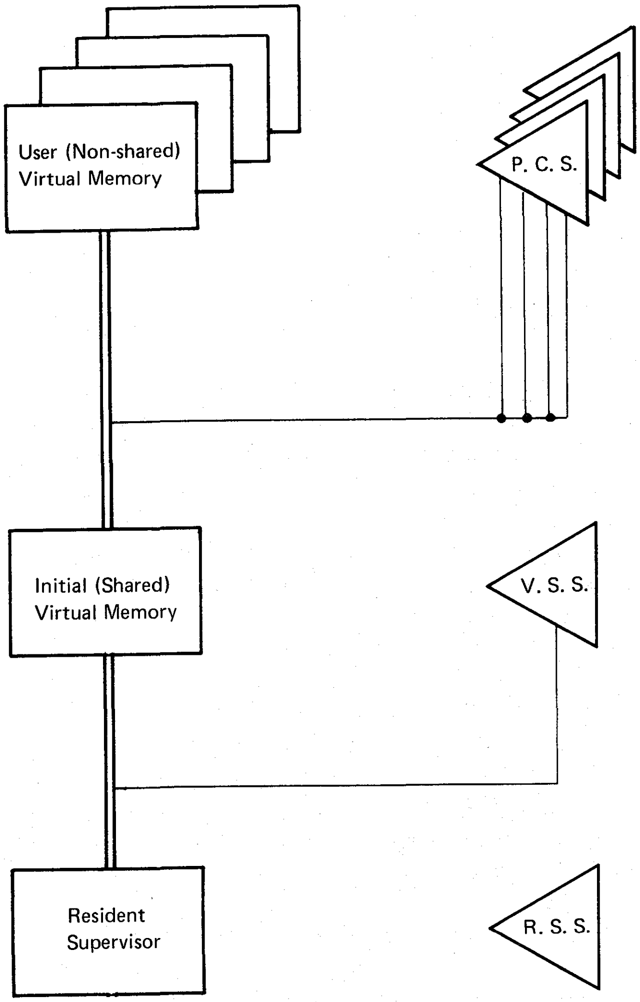

User (Non-shared) Virtual Memory
P. C. S.
Initial (Shared) Virtual Memory
V. S. S.
Resident Supervisor
R. S. S.

Figure 6—Schematic views of TSS/360 debugging aids

The Time-Sharing Support System (TS^3) which was delivered with Release 3.0 removed the final dependency upon STRATO. The Resident Support Subsystem (RSS) and Virtual Support Subsystem (VSS) are more general and easier to use than STRATO command language, but they have one significant weakness. RSS itself is dependent upon some resident supervisor facilities, e.g., interrupt stacker and supervisor core allocation. The effect is that RSS is not a valuable debugging aid for those parts of the resident supervisor.

One of the most valuable aids in TSS was a fortuitous invention—dynamic modification of the system through the use of Delta Data Sets. The system initialization program, STARTUP, builds both the resident supervisor and Initial Virtual Memory (IVM) from modules stored on the Initial Program Load (IPL) Control Volume. (This is in contrast to OS/360 in which the entire supervisor is link edited together during the System Generation procedure.) If dynamic modification of the resident supervisor or IVM is desired, the changes are prepared on a private disk and the operator informs STARTUP to search this disk for "deltas." The process is shown schematically in Figure 5. The modifications included in this manner stay in effect only until shutdown. The next time the system is started, they can be removed or altered at the user's option.

To understand the next phase in the development of debugging aids, let us summarize where we stood with Release 3.0. Figure 6 gives a schematic view of the system operation. At his individual terminal, each user had at his command a comprehensive set of debugging aids to work within his virtual memory. With the **P**rogram **C**ontrol **S**ubsystem (PCS) of Command System II, he could dynamically start, stop, or interrupt programs, and he could display or patch within them. The TSS Dynamic Loader allowed him to substitute an altered copy for any module within his virtual memory. This power and flexibility was replicated for each active user, i.e., debugging in non-shared virtual memory was time-shared.

The same was not true either for shared virtual memory or the resident supervisor. Exploratory debugging could be done with RSS and VSS but modules could only be replaced by going through STARTUP. What was desired beyond these aids were techniques to allow time-shared debugging work on both shared virtual memory and the resident supervisor.

Such a facility for modules within shared virtual memory is provided by HOOK. With HOOK a system programmer can cause a private (modified) copy of a shared virtual memory module to be used *for his task only*. Linkage to the private copy is established dynamically in response to a HK command. If testing shows an error in the module, the programmer can UNHOOK it, assemble further modifications, and HOOK that version for further testing. This activity goes on in parallel with other use of the system, including other users of HOOK. What was said previously about the value of measurement tools to users is probably more true of debugging aids. Recognizing this and realizing that users desire to modify shared as well as non-shared virtual memory, HOOK was made available to all TSS users late in 1971.

Achieving time-shared debugging for the resident supervisor is a more difficult challenge. To meet it, TSS Development looked outside its own domain to Virtual Machines. Such technology underlies the design of CP-67/CMS.[8] CP-67 creates a complete Virtual Machine for each user, so several users can work with virtual Model 67 machines and debug versions of CP. It is clearly feasible to provide Virtual Machines under TSS, and the addition of such a facility would complete a most powerful set of program debugging aids. With Virtual Machines, there would be little, if any, need for single-thread, restart-prone debugging. Figure 7 shows the debugging aids as they could be with Virtual Machine added.

MANAGING SOFTWARE DEVELOPMENT

Bringing any large system from concept to fruition is a difficult task. TSS Development had many problems, some not unlike those described recently by F. P. Brooks, Jr.[9] Probably the most valuable lesson learned in the process was how to manage a large software development project in the "steady state."

The TSS development organization was composed of three major functions: Design, Development, and Build and Test.[10] Ideally, these functions should be accomplished serially, for in this case, the same people could do the work, and no need would exist for communications mechanisms between functions. For practical purposes, the functions do overlap. The design cannot be complete or perfect. The development job must continue to correct program errors and implement new functions. Build and Test is required until the total system is stabilized. Further, the size of TSS both in concept and design precluded implementation by a small group.

Here I will insert an opinion on the merits of large versus small development groups. A small group has two distinct advantages. It is easy to manage (one

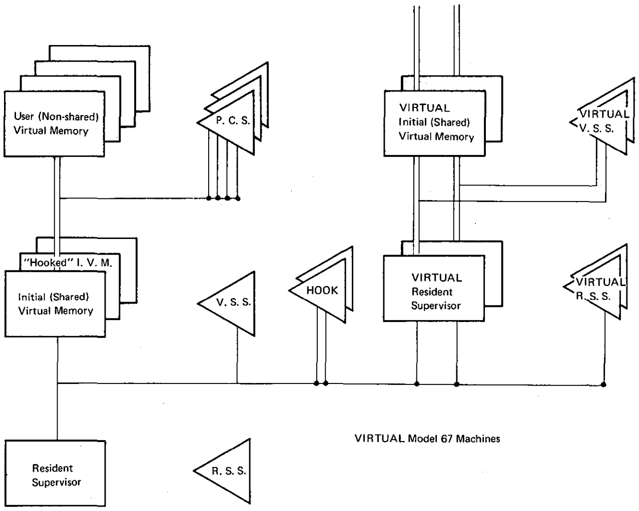


Figure 7—Schematic views of TSS/360 debugging aids

person can direct the efforts of 7-12 people); and a uniformly high standard of quality can be maintained in the selection of the group's members. Large groups suffer from the opposite problems. There is, however, nothing inherently bad about a large programming group. The critical requirement is for effective management.

The systems management technique employed in TSS was known as the New Scope Review Board (NSRB). Control was exercised in the following way. The design of TSS was considered complete at the Release 1.0 level so that the only source of new code would be fixes to identified programming errors (APAR's). Any other change that was identified was designed and presented to the NSRB for approval. The board was composed of representatives of all functional areas with the System Manager serving as chairman. Approval of a given item meant inclusion of its design into the system and hence implementation, test, and integration in a System Release. With this mechanism in place, sources of new code were restricted to error correction and approved NSRB items.

Each proposal to change TSS was described in an NSRB Technical Summary, composed of eight (8) parts.

1. Title of the item and sequence number
2. Four line abstract of the item
3. Reference to the design document
4. List of all external publications affected
5. A description of the character of the change, e.g., does the item affect performance, reliability, human factors, or function—positively or negatively—and how much?
6. List of other items upon which this item depends, i.e., other NSRB items, APAR's or hardware Engineering Changes
7. List of all parts of the system affected, e.g., what modules, DSECT's, Macros, or Data Sets are revised, added or deleted—and, if a revision, is the change small, medium, or large?
8. Signature of the designer and his manager

With all proposals for new programming before the Board and in a standard format, the system became manageable. For instance, a concentrated program to improve performance was instituted by selecting NSRB items whose character indicated large, performance plus while rejecting others. A control on resources was also possible since each NSRB item carried with it an estimate of the work required to complete it. The System Manager could either restrict the quantity of new scope to match his resource or he could add or substract resource to match the new scope desired.

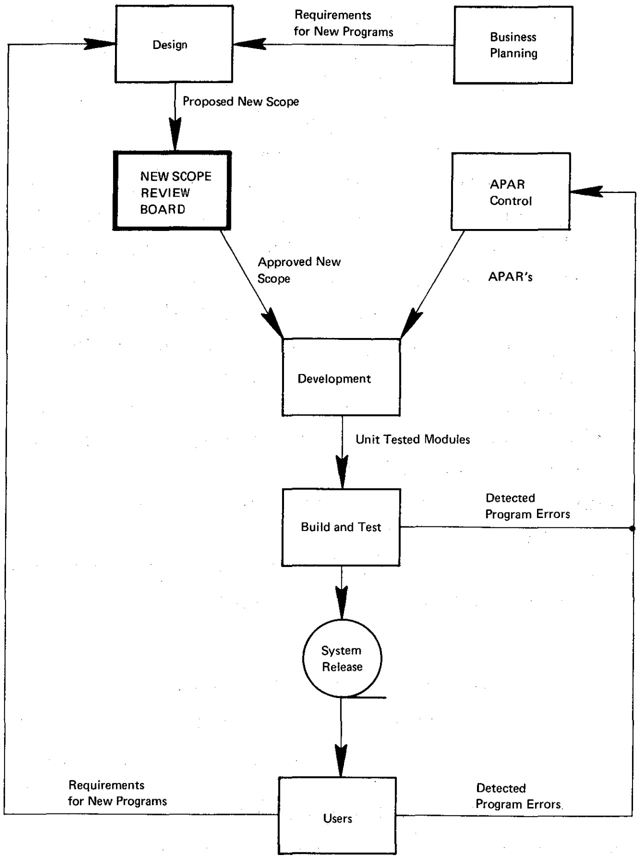


Figure 8—TSS/360 development cycle showing major functions and the flow of information

Figure 8 illustrates the interrelationships of the various functions and the flow of information between them.

The major sources for requirements input are the System Users and a Business Planning Group. Once a requirement is known, a design is prepared and the item is submitted to the NSRB. Those items which are approved form one of two sources of work for the development group.

Here a word should be said about the unique place occupied by the design function. We have mentioned that TSS had a sound basic design, and certainly this was essential to the success of its implementation. We also subscribe to the philosophy that designers must have control over the manner in which the system is developed. Yet, design must be responsive to a multitude of outside influences, including Systems Management. Several of the best features of TSS have resulted from corrections to short-sited design.

A philosophy used throughout TSS has been to deliver the best possible code. Therefore, all program errors detected by the build and test group are submitted to the same APAR Control group as User-detected errors. APAR control verifies the error, eliminates duplicates, and tracks the status of all errors. Errors to be fixed form the second source of work for Development. Here we have had success with the idea of module ownership. The development programmer responsible for a module does all the work necessary to implement NSRB changes and APAR corrections. (Having a separate maintenance group tends to remove an incentive for qualtity development work, i.e., ones bugs come back to haunt the maintenance programmer not the developer.) The resultant changed modules are the source of input to Build and Test.

The cycle is completed when Build and Test ships a system release.

The table below shows the activity of TSS Development from the three sources described.

TABLE II—System Content by Release

Release	Number of NSRB Items Included	Number of APARS Corrected	
		User-detected	Internal
1.1	9	91	—
1.2	32	136	7
2.0	49	181	72
3.0	56	190	255
4.0	42	111	201
5.0	80	321	534
5.1	32	192	471
6.0	84	226	538
6.1	—	40	85
7.0	64	174	466
8.0	50	206	442
8.1	70	574	1016
Subsequent	—	NA	NA
TOTAL	568	2442	4087

CONCLUSION

Where does TSS go from here? What is the prognosis for large-scale, interactive systems in general? The first question is easier to answer.

Work remains to be done to bring TSS up to a high degree of reliability. This task is receiving the highest priority. In addition, we are dedicated to improving the system's error recovery facilities. There is a high potential plateau for dependability and availability; we are working to reach it.

Impressive as the TSS performance story has been, we accept our users' contention that performance is not as good as it needs to be. The proper course now is to specify new benchmarks that more closely parallel the real customers workloads, and to use the powerful tools previously described to drive performance upward.

Last, but not least, is the fact that TSS users have been and are continuing to improve the system. Both the usability and the utility of TSS continue to grow.

The prognosis? I cannot provide that, but I can put forward two thoughts—both deeply rooted in the TSS/360 experience.

I have noted that where TSS/360 has succeeded people efficiency is held more important than machine efficiency, and that it is becoming broadly accepted that the availability of skilled people, not the availability of better hardware, is the limiting factor in the growth of our industry. From this I conclude that, if our industry is to thrive, systems which go to the user, which make people more efficient, in short, interactive systems like TSS must play a dominant role in the decade ahead.

Finally, I will express my belief that the single most important requirement before us is to modernize the program development process itself. Years ago, the application of computing to hardware design and manufacture removed bottlenecks which threatened to retard industry growth. Today, the bottlenecks are in software design and manufacture.

ACKNOWLEDGMENTS

The outlook of this paper is historical not journalistic. I hope that readers who have been more deeply involved in TSS development than I will agree with the sense of the paper even though they disagree with this or that detail.

I have cited some references, and I have made use of some unpublished information. For the latter, I would like to thank M. R. Babacci, D. R. Cease, K. M. Hilmar, A. Kamerman, O. R. LaMaire, and C. E. Seabold.

Finally, it must be said that TSS/360 would have been a very uninteresting subject were it not the creation of fine people. Each man and woman who worked on the system deserves a fair share of credit. My special thanks go to Bill Florac, Scott Locken, Nick Martellotto, and Ida Scott. These people more than any others gave me insight into the nature of TSS.

REFERENCES

1 W T COMFORT
A computing system design for user service
1965 FJCC Sparten Books

2 A S LETT W L KONIGSFORD
TSS/360: A time-shared operating system
1968 FJCC Thompson Book Company

3 *IBM System/360 time sharing system—Command system users guide*
IBM Corporation Form GC28–2001

4 W J DOHERTY
Scheduling TSS/360 for responsiveness
1970 FJCC AFIPS Press

5 D N STREETER
Cost/benefits of computing services in a scientific environment
IBM Research Report RC 3453

6 F D SCHULMAN
Hardware measurement device for IBM System/360 time sharing evaluation
22nd National ACM Conference Proceedings

7 W R DENISTON
SIPE: A TSS/360 software measurement technique
24th National ACM Conference Proceedings

8 *CP–67/CMS system description manual*
IBM Form GH20–0802

9 F P BROOKS JR
Why is the software late?
Data Management August 1971

10 N A ZIMMERMAN
System integration as a programming function
24th National ACM Conference Proceedings

Multics—The first seven years*

by F. J. CORBATÓ and J. H. SALTZER

Massachusetts Institute of Technology
Cambridge, Massachusetts

and

C. T. CLINGEN

Honeywell Information Systems Inc.
Cambridge, Massachusetts

INTRODUCTION

In 1964, following implementation of the Compatible Time-Sharing System (CTSS)[1,2] serious planning began on the development of a new computer system specifically organized as a prototype of a computer utility. The plans and aspirations for this system, called Multics (for **Multi**plexed **I**nformation and **C**omputing **S**ervice), were described in a set of six papers presented at the 1965 Fall Joint Computer Conference.[3-8] The development of the system was undertaken as a cooperative effort involving the Bell Telephone Laboratories (from 1965 to 1969), the computer department of the General Electric Company,* and Project MAC of M.I.T.

Implicit in the 1965 papers was the expectation that there should be a later examination of the development effort. From the present vantage point, however, it is clear that a definitive examination cannot be presented in a single paper. As a result, the present paper discusses only some of the many possible topics. First we review the goals, history and current status of the Multics project. This review is followed by a brief description of the appearance of the Multics system to its various classes of users. Finally several topics are given which represent some of the research insights which have come out of the development activities. This organization has been chosen in order to emphasize those aspects of software systems having the goals of a computer utility which we feel to be of special interest. We do not attempt detailed discussion of the organization of Multics; that is the purpose of specialized technical books and papers.*

GOALS

The goals of the computer utility, although stated at length in the 1965 papers, deserve a brief review. By a computer utility it was meant that one had a community computer facility with:

(1) Convenient remote terminal access as the normal mode of system usage;
(2) A view of continuous operation analogous to that of the electric power and telephone companies;
(3) A wide range of capacity to allow growth or contraction without either system or user reorganization;
(4) An internal file system so reliable that users trust their only copy of programs and data to be stored in it;
(5) Sufficient control of access to allow selective sharing of information;
(6) The ability to structure hierarchically both the logical storage of information as well as the administration of the system;
(7) The capability of serving large and small users without inefficiency to either;
(8) The ability to support different programming environments and human interfaces within a single system;

* Work reported herein was sponsored (in part) by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under office of Naval Research Contract Number N00014–70–A–0362–0001. Reproduction is permitted for any purpose of the United States Government.

* Subsequently acquired by Honeywell Information Systems Inc.

* For example, the essential mechanisms for much of the Multics system are given in books by Organick[9] and Watson.[10]

(9) The flexibility and generality of system organization required for evolution through successive waves of technological improvements and the inevitable growth of user expectations.

In an absolute sense the above goals are extremely difficult to achieve. Nevertheless, it is our belief that Multics, as it now exists, has made substantial progress toward achieving each of the nine goals.* Most importantly, none of these goals had to be compromised in any important way.

HISTORY OF THE DEVELOPMENT

As previously mentioned, the Multics project got under way in the Fall of 1964. The computer equipment to be used was a modified General Electric 635 which was later named the 645. The most significant changes made were in the processor addressing and access control logic where paging and segmentation were introduced. A completely new Generalized Input/Output Controller was designed and implemented to accommodate the varied needs of devices such as disks, tapes and teletypewriters without presenting an excessive interrupt burden to the processors. To handle the expected paging traffic, a 4-million word (36-bit) high-performance drum system with hardware queueing was developed. The design specifications for these items were completed by Fall 1965, and the equipment became available for software development in early 1967.

Software preparation underwent several phases. The first phase was the development and blocking out of major ideas, followed by the writing of detailed program module specifications. The resulting 3,000 typewritten pages formed the Multics System Programmers' Manual and served as the starting point for all programming. Furthermore, the software designers were expected to implement their own designs. As a general policy PL/I was used as the system programming language wherever possible to maximize lucidity and maintainability of the system.[14,15] This policy also increased the effectiveness of system programmers by allowing each one to keep more of the system within his grasp.

The second major phase of software development, well under way by early 1967, was that of module implementation and unit checkout followed by merging into larger aggregates for integrated testing. Up to then most software and hardware difficulties had been anticipated on the basis of previous experience. But what gradually became apparent as the module integration continued was that there were gross discrepancies between actual and expected performance of the various logical execution paths throughout the software. The result was that an unanticipated phase of design iterations was necessary. These design iterations did not mean that major portions of the system were scrapped without being used. On the contrary, until their replacements could be implemented, often months later, they were crucially necessary to allow the testing and evaluation of the other portions of the system. The cause of the required redesigns was rarely "bad coding" since most of the system programmers were well above average ability. Moreover the redesigns did not mean that the goals of the project were compromised. Rather three recurrent phenomena were observed: (1) typically, specifications representing less-important features were found to be introducing much of the complexity, (2) the initial choice of modularity and interfacing between modules was sometimes awkward and (3) it was rediscovered that the most important property of algorithms is simplicity rather than special mechanisms for unusual cases.*

The reason for bringing out in detail the above design iteration experience is that frequently the planning of large software projects still does not properly take the need for continuing iteration into account. And yet we believe that design iterations are a required activity on any large scale system which attempts to break new conceptual ground such that individual programmers cannot comprehend the entire system in detail. For when new ground is broken, it is usually impossible to deduce the consequent system behavior except by experimental operation. Simulation is not particularly effective when the system concepts and user behavior are new. Unfortunately, one does not understand the system well enough to simplify it correctly and thereby obtain a manageable model which requires less effort to implement than the system itself. Instead one must develop a different view:

(1) The initial program version of a module should be viewed only as the first complete specification of the module and should be subject to design review *before* being debugged or checked out.

(2) Module design and implementation should be based upon an assumption of periodic evaluation, redesign, and evolution.

In retrospect, the design iteration effect was apparent

* To the best of our knowledge, the only other attempt to comprehensively attack all of these goals simultaneously is the TSS/360 project at IBM.[11,12,13]

* "In anything at all, perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away . . ."

—Antoine de Saint-Exupéry, *Wind, Sand and Stars* Quoted with permission of Harcourt Brace Jovanovich, Inc.

even in the development of the earlier Compatible Time-Sharing System (CTSS) when a second file system with many functional improvements turned out to have poor performance when initially installed. A hasty design iteration succeeded in rectifying the matter but the episode at the time was viewed as an anomaly perhaps due to inadequate technical review of individual programming efforts.

CURRENT STATUS

In spite of the unexpected design iteration phase, the Multics system became sufficiently effective by late 1968 to allow system programmers to use the system while still developing it. By October 1969, the system was made available for general use on a "cost-recovery" charging basis similar to that used for other major computation facilities at M.I.T. Multics is now the most widely used time-sharing system at M.I.T., supporting a user community of some 500 registered subscribers. The system is currently operated for users 22 hours per day, 7 days per week. For at least eight hours each day the system operates with two processors and three memory modules containing a total of 384k (k = 1024) 36-bit words. This configuration currently is rated at a capacity of about 55 fairly demanding users such that most trivial requests obtain response in one to five seconds. (Future design iterations are expected to increase the capacity rating.) Several times a day during the off-peak usage hours the system is dynamically reconfigured into two systems: a reduced capacity service system and an independent development system. The development system is used for testing those hardware and software changes which cannot be done under normal service operation.

The reliability of the round-the-clock system operation described above has been a matter of great concern, for in any on-line real-time system the impact of mishaps is usually far more severe than in batch processing systems. In an on-line system especially important considerations are:

(1) the time required before the system is usable again following a mishap,
(2) the extra precautions required for restoring possibly lost files, and
(3) the psychological stress of breaking the interactive dialogue with users who were counting on system availability.

Because of the importance of these considerations, careful logs are kept of all Multics "crashes" (i.e., system service disruption for all active users) at M.I.T. in order that analysis can reveal their causes. These analyses indicate currently an average of between one and two crashes per 24 hour day. These crashes have no single cause. Some are due to hardware failures, others to operator error and still others to software bugs introduced during the course of development. At the two other sites where Multics is operated, but where active system development does not take place, there have been almost no system failures traced to software.

TABLE I—A comparison of the system development and use periods of CTSS and Multics. The Multics development period is not significantly longer than that for CTSS despite the development of about 10 times as much code for Multics as for CTSS and a geographically distributed staff. Although reasons for this similarity in time span include the use of a higher-level programming language and a somewhat larger staff, the use of CTSS as a develoрment tool for Multics was of pivitol importance.

System	*Development Only*	*Development + Use*	*Use Only*
CTSS	1960-1963	1963-1965	1965-present
Multics	1964-1969	1969-present	

Currently the Multics system, including compilers, commands, and subroutine libraries, consists of about 1500 modules, averaging roughly 200 lines of PL/I apiece. These compile to produce some 1,000,000 words of procedure code. Another measure of the system is the size of the resident supervisor which is about 30k words of procedure and, for a 55 user load, about 36k words of data and buffer areas.

Because the system is so large, the most powerful maintenance tool available was chosen—the system itself. With all of the system modules stored on-line, it is easy to manipulate the many components of different versions of the system. Thus it has been possible to maintain steadily for the last year or so a pace of installing 5 or 10 new or modified system modules a day. Some three-quarters of these changes can be installed while the system is in operation. The remainder, pertaining to the central supervisor, are installed in batches once or twice a week. This on-line maintenance capability has proven indispensable to the rapid development and maintenance of Multics since it permits constant upgrading of the user interface without interrupting the service. We are just beginning to see instances of user-written applications which require this same capability so that the application users need not be interrupted while the software they are using is being modified.

The software effort which has been spent on Multics is difficult to est:mate. Approximately 150 man-years were applied directly to design and system programming during the "development-only" period of Table I.

Since then we estimate that another 50 man-years have been devoted to improving and extending the system. But the actual cost of a single successful system is misleading, for if one starts afresh to build a similar system, one must compensate for the non-zero probability of failure.

THE APPEARANCE OF MULTICS TO ITS USERS

Having reviewed the background of the project, we may now ask who are the users of the Multics system and what do the facilities that Multics provides mean to these users. Before answering, it is worth describing the generic user as "viewed" by Multics. Although from the system's point of view all users have the same general characteristics and interface with it uniformly, no single human interface represents the Multics machine. That machine is determined by each user's initial procedure coupled with those functions accessible to him. Thus there exists the potential to present each Multics user with a unique external interface.

However, Multics does provide a native internal program environment consisting of a stack-oriented, pure-procedure, collection of PL/I procedures imbedded in a segmented virtual memory containing all procedures and data stored on-line. The extent to which some, all, or none of this internal environment is visible to the various users is an administrative choice.

The implications of these two views—both the external interface and the internal programming environment—are discussed in terms of the following categories of users:

- System programmers and user application programmers responsible for writing system and user software.
- Administrative personnel responsible for the management of system resources and privileges.
- The ultimate users of applications systems.
- Operations and hardware maintenance personnel responsible, respectively, for running the machine room and maintaining the hardware.

Multics as viewed by system and subsystem programmers

The machine presented to both the Multics system programmer and the application system programmer is the one with which we have the most experience; it is the raw material from which one constructs other environments. It is worth reemphasizing that the only differentiation between Multics system programmers and user programmers is embodied in the access control mechanism which determines what on-line information can be referenced; therefore, what are apparently two groups of users can be discussed as one.

Major interfaces presented to programmers on the Multics system can be classified as the program preparation and documentation facilities and the program execution and debugging environment. They will be touched upon briefly, in the order used for program preparation.

Program preparation and documentation

The facilities for program preparation on Multics are typical of those found on other time-sharing systems, with some shifts in emphasis (see the Appendix). For example, programmers consider the file system sufficiently invulnerable to physical loss that it is used casually and routinely to save all information. Thus, the punched card has vanished from the work routine of Multics programmers and access to one's programs and the ability to work on them are provided by the closest terminal.

As another example, the full ASCII character set is employed in preparing programs, data, and documentation, thereby eliminating the need for multiple text editors, several varieties of text formatting and comparison programs, and multiple facilities for printing information both on-line and off-line. This generalization of user interfaces facilitates the learning and subsequent use of the system by reducing the number of conventions which must be mastered.

Finally, because the PL/I compiler is a large set of programs, considerable attention was given to shielding the user from the size of the compiler and to aiding him in mastering the complexities of the language. As in many other time-sharing systems, the compiler is invoked by issuing a simple command line from a terminal exactly as for the less ambitious commands. No knowledge is required of the user regarding the various phases of compilation, temporary files required, and optional capabilities for the specialist; explanatory "sermons" diagnosing syntactic errors are delivered to the terminal to effect a self-teaching session during each compilation. To the programmer, the PL/I compiler is just another command.

Program execution environment

Another set of interfaces is embodied in the implementation environment seen by PL/I programmers. This environment consists of a directly addressable virtual memory containing the entire hierarchy of on-line information, a dynamic linking facility which searches this hierarchy to bind procedure references, a device-independent input/output[16] system,* and program debugging and metering facilities. These facilities enjoy a symbiotic relationship with the PL/I procedure environment used both to implement them and to implement user facilities co-existing with them. Of major significance is that the natural internal environment provided and required by the system is exactly that environment expected by PL/I procedures. For example, PL/I pointer variables, call and return statements, conditions, and static and automatic storage all correspond directly to mechanisms provided in the internal environment. Consequently, the system supports PL/I code as a matter of course.

The main effect of the combination of these features is to permit the implementer to spend his time concentrating on the logic of his problem; for the most part he is freed from the usual mechanical problems of storage management and overlays, input/output device quirks, and machine-dependent features.

Some implementation experience

The Multics team began to be much more productive once the Multics system became useful for software development. A few cases are worth citing to illustrate the effectiveness of the implementation environment. A good example is the current PL/I compiler, which is the third one to be implemented for the project, and which consists of some 250 procedures and about 125k words of object code. Four people implemented this compiler in two years, from start to first general use. The first version of the Multics program debugging system, composed of over 3,000 lines of source code, was usable after one person spent some six months of nights and weekends "bootlegging" its implementation. As a last example, a facility consisting of 50 procedures with a total of nearly 4,000 PL/I statements permitting execution of Honeywell 635 programs under Multics became operational after one person spent eight months learning about the GCOS operating system for the 635, PL/I, and Multics, and then implemented the environment. In each of these examples the implementation was accomplished from remote terminals using PL/I.

Multics users have discovered that it is possible to get their programs running very quickly in this environment. They frequently prepare "rough drafts" of programs, execute them, and then improve their overall design and operating strategy using the results of experience obtained during actual operation. As an example, again drawn from the implementation of Multics, the early designs and implementations of the programs supporting the virtual memory[18] made overoptimistic use of variable-sized storage allocation techniques. The result was a functionally correct but inadequately performing set of programs. Nevertheless, these modules were used as the foundation for subsequent work for many months. When they were finally replaced with modules using simplified fixed-size storage techniques, performance improvements of over an order of magnitude were realized. This technique emphasizes two points: first, it is frequently possible to provide a practical, usable facility containing temporary versions of programs; second, often the insight required to significantly improve the behavior of a program comes only after it is studied in operation. As implied in the earlier discussion of design iteration, our experience has been that structural and strategic changes rather than "polishing" (or recoding in assembly language) produce the most significant performance improvements.

In general, we have noticed a significant "amplifier" or "leverage" effect with the use of an effective on-line environment as a system programming facility. Major implementation projects on the Multics system seldom involve more than a few programmers, thereby easing the management and communications problems usually entailed by complex system implementations. As would be expected, the amplification effect is most apparent with the best project personnel.

Administration of Multics facilities and resources

The problem of managing the capabilities of a computer utility with geographically dispersed subscribers leads to a requirement of decentralized administration. At the apex of an administrative pyramid resides a system administrator with the ability to register new users, confer resource quotas, and generate periodic bills for services rendered. The system administrator deals with user groups called projects. Each group can in turn designate a project administrator who is delegated the authority to manage a budget of system resources on behalf of the project. The project administrator is then free to deal directly with project members without further intervention from the system administrator, thereby greatly reducing the bottlenecks inherent in a completely centralized administrative structure.

Environment shaping

In addition to having immediate control of such resources as secondary storage, port access, and rate of processor usage, the project administrator is also able to define or shape the environment seen by the members

* The Michigan Terminal System[17] has a similar device-independent input/output system.

of his project when they log into the system. He does this by defining those procedures that can be accessed by members of his project and by specifying the initial procedure executed by each member of his project when he logs in. This environment shaping facility has led to the notion of a private project subsystem on Multics. It combines the administrative and programming facilities of Multics so that a project administrator and a few project implementers can build, maintain, and evolve environments entirely on their own. Thus, some subsystems bear no internal resemblance to the standard Multics procedure environment.

For example, the Dartmouth BASIC[19] compiler executes in a closed subsystem implemented by an M.I.T. student group for use by undergraduate students. The compiler, its object code, and all support routines execute in a simulation of the native environment provided at Dartmouth. The users of this subsystem need little, if any, knowledge of Multics and are able to behave as if logged into the Dartmouth system proper. Other examples of controlled environment subsystems include one to permit many programs which normally run under the GCOS operating system to also run unmodified in Multics. Finally, an APL[20] subsystem allows the user to behave for the most part as if he were logged into an APL machine. The significance of these subsystems is that their implementers did not need to interact with the system administrator or to modify already existing Multics capabilities. The administrative facilities permit each such subsystem to be offered by its supporters as a private service with its own group of users, each effectively having its own private computer system.

Other Multics users

Finally, we observe that the roles of the application user, the system operators and the hardware maintainers as seen by the system are simply those of ordinary Multics users with specialized access to the on-line procedures and data. The effect of this uniformity of treatment is to reduce greatly the maintenance burden of the system control software. One example, of great practical importance, has been the ease with which system performance measurement tools have been prepared for use by the operating staff.

INSIGHTS

So far, we have discussed the status and appearance of the Multics system. A further question is what has been learned in the construction of Multics which is of use to the designers of other systems. Having a bright idea which clearly solves a problem is not sufficient cause to claim a contribution if the idea is to be part of a complex system. In order to establish the real feasibility of an idea, all of its implications and consequences must be followed out. Much of the work on Multics since 1965 has involved following out implications and consequences of the many ideas then proposed for the prototype computer utility. That following out is an essential part of proof of ideas is attested by the difficulties which have been encountered in other engineering efforts such as the development of nuclear fusion power plants and the electric automobile. Not all proposals work out; for example, extended attempts to engineer an atomic powered airplane suggest infeasibility.

Perhaps Multics' most significant single contribution to the state of the art of computer system construction is the demonstration of a large set of fully implemented ideas in a working system. Further, most of these ideas have been integrated without straining the overall design; most additional proposals would not topple the structure. Ideas such as virtual memory access to online storage, parallel process organization, routine but controlled information sharing, dynamic linking of procedures, and high-level language implementa-

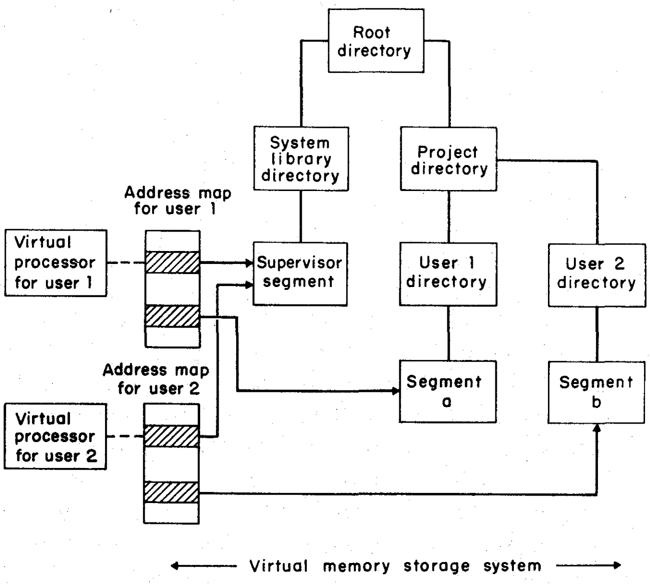


Figure 1—The entire storage hierarchy may be mapped into individual user process address spaces (see arrows) as if contained in primary memory. Illustrated are the sharing of a supervisor segment by user 1 and user 2 and private access to segment a and segment b. The necessary primary storage is simulated by a demand paging technique which moves information between the real primary memory and secondary storage

tion have proven remarkably compatible and complementary.

To illustrate some of the areas of progress in understanding of system organization and construction which have been achieved in Multics, we consider here the following five topics:

1. Modular division of responsibility
2. Dynamic reconfiguration
3. Automatically managed multilevel memory
4. Protection of programs and data
5. System programming language

Modular division of responsibility

Early in the design of Multics a decision had to be made whether or not to treat the segmented virtual memory as a separately usable "feature," independent of a traditionally organized read/write type file system. The alternative, to use the segmented virtual memory as the file system itself, providing the illusion of direct "in-core" access to all on-line storage, was certainly the less conservative approach (see Figure 1). The second approach, which was the one chosen, led to a strong test of the ability of a computing system to support an apparent one-level memory for an arbitrarily large information base. It is interesting that the resulting almost total decoupling between physical storage allocation and data movement on the one hand and directory structure, naming, and file organization on the other led to a remarkably simple and functionally modular structure for that part of the system[18] (see Figure 2).

Another area of Multics in which a high degree of functional modularity was achieved was in the area of scheduling, multiprogramming, and processor management. Because harnessing of multiple processors was an objective from the beginning, a careful and methodical approach to multiplexing processors, handling interrupts, and providing interprocess synchronizing primitives was developed. The resulting design, known as the Multics traffic controller, absorbed into a single, simple module a set of responsibilities often diffused among a scheduling algorithm, the input/output controlling system, the on-line file management system, and special purpose inter-user communication mechanisms.[21]

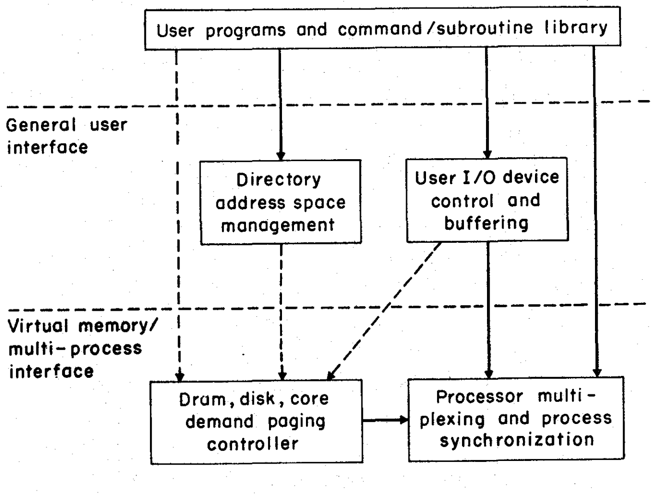

User programs and command/subroutine library
General user interface
Directory address space management
User I/O device control and buffering
Virtual memory/ multi-process interface
Drum, disk, core demand paging controller
Processor multiplexing and process synchronization

Figure 2—Major lines of modular division in Multics. Solid lines indicate calls for services. Dotted lines indicate implicit use of the virtual memory

Finally, with processor management and on-line storage management uncoupled into well-isolated modules, the Multics input/output system was left with the similarly isolatable function of managing streams of data flowing from and to source and sink type devices.[16] Thus, this section of the system concentrates only on switching of the streams, allocation of data buffering areas, and device control strategies.

Each of the divisions of labor described above represents an interesting result primarily because it is so difficult to discover appropriate divisions of complex systems.* Establishing that a certain proposed division results in simplicity, creates an uncluttered interface, and does not interfere with performance, is generally cause for a minor celebration.

Dynamic reconfiguration

If the computer utility is ever to become as much a reality as the electric power utility or the telephone communication service, its continued operation must not be dependent upon any single physical component, since individual components will eventually require maintenance. This observation leads an electric power utility to provide procedures whereby an idle generator may be dynamically added to the utility's generating capacity, while another is removed for maintenance, all without any disruption of service to customers. A similar scenario has long been proposed for multiprocessor, multimemory computer systems, in which one would dynamically switch processsors and memory boxes in and out of the operating configuration as needed. Unfortunately, though there have been demonstrated a few "special purpose" designs,* it has not been apparent how to provide for such operations in a general purpose system. A recent thesis[24] proposed a general model for the dynamic binding and unbinding of computation and memory structures to and from ongoing computa-

* See Dijkstra[22] for a further discussion of this point.

* An outstanding example is the American Airlines SABRE system.[23]

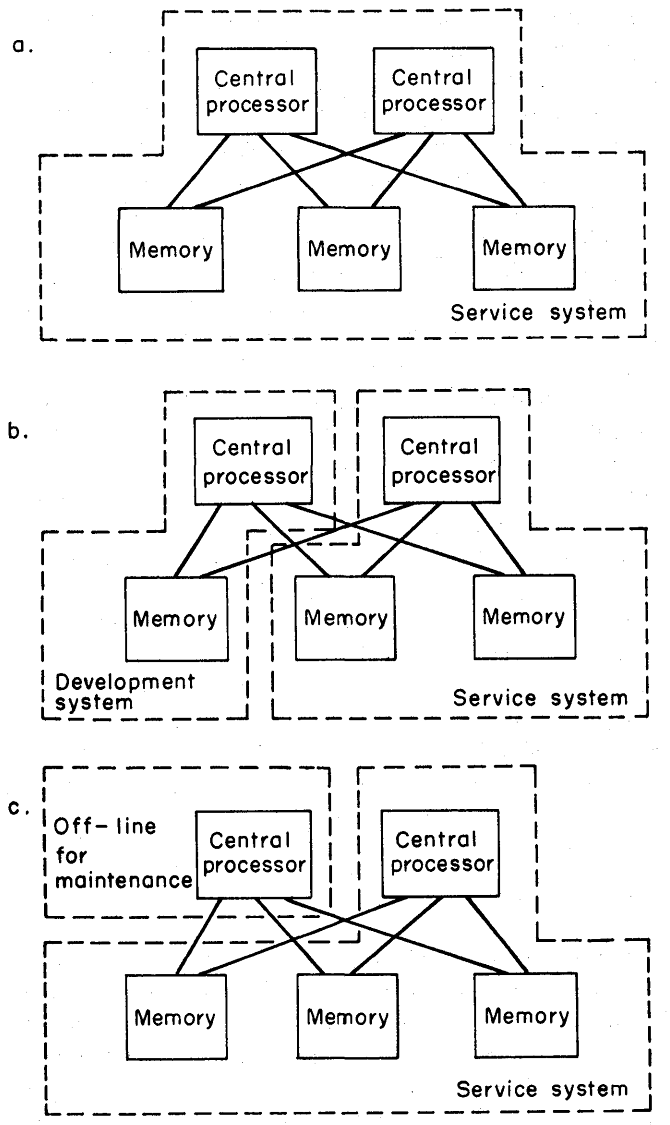


Figure 3—Dynamic reconfiguration permits switching among the three typical operating configurations shown here, without currently logged-in users being aware that a change has taken place

tions. Using this model as a basis, the thesis also proposed a specific implementation for a typical multiprocessor, multimemory computing system. One of the results of this work was the addition to the operating Multics system of the capability of dynamically adding and removing central processors and memory modules as in Figure 3. The usefulness of the idea may be gauged by observing that at M.I.T. five to ten such reconfigurations are performed in a typical 24-hour operating day. Most of the reconfigurations are used to provide a secondary system for Multics development.

Automatically managed multilevel memory

By now it has become accepted lore in the computer system field that the use of automatic management algorithms for memory systems constructed of several levels with different access times can provide a significant reduction of user programming effort. Examples of such automatic management strategies include the buffer memories of the IBM system 370 models 155, 165, and 195[25] and the demand paging virtual memories of Multics, IBM's CP-67[26] and the Michigan Terminal System.[17] Unfortunately, behind the mask of acceptance hides a worrisome lack of knowledge about how to engineer a multilevel memory system with appropriate strategy algorithms which are matched to the load and hardware characteristics. One of the goals of the Multics project has been to instrument and experiment with the multilevel memory system of Multics, in order to learn better how to predict in advance the performance of proposed new automatically managed multilevel memory systems. Several specific aspects of this goal have been explored:

- A strategy to treat core memory, drum, and disk as a three-level system has been proposed, including a "least-recently-used" algorithm for moving information from drum to disk. Such an algorithm has been used for some time to determine which pages should be removed from core memory.[27] The dynamics of interaction among two such algorithms operating at different levels are weakly understood, and some experimental work should provide much insight. The proposed strategy will be implemented, and then compared with the simpler present strategy which never moves things from drum to disk, but instead makes educated "guesses" as to which device is most appropriate for the permanent residence of a given page. If the automatic algorithm is at least as good as the older, static one, it would represent an improvement in overall design by itself, since it would automatically track changes in user behavior, while the static algorithm requires attention to the validity of its guesses.
- A scheme to permit experimentation with predictive paging algorithms was devised. The scheme provides for each process a list of pages to be preloaded whenever the process is run, and a second list to be immediately purged whenever the process stops. The updating of these lists is controlled by a decision table exercised every time the process stops running. Since every page of the Multics virtual memory is potentially shared, the decision table represents a set of heuristics designed to separate out those which are probably not being shared at the moment.

- A series of measurements was made to establish the effectiveness of a small hardware associative memory used to hold recently accessed page descriptors. These measurements established a profile of hit ratio (probability of finding a page descriptor in the associative memory) versus associative memory size which should be useful to the designers of virtual memory systems.[28]
- A set of models, both analytic and simulation, was constructed to try to understand program behavior in a virtual memory. So far, two results have been obtained. One is the finding that a single program characteristic (the mean execution time before encountering a "missing" page in the virtual memory as a function of memory size) suffices to provide a quite accurate prediction of paging and idle overheads. The second is direct calculation of the distribution of response times under multiprogramming. Having available the entire response time distribution, rather than just averages, permits estimation of the variance and 90-percentile points of the distribution, which may be more meaningful than just the average. A doctoral thesis is in progress on this topic.

Although the immediate effect of each of these investigations is to improve the understanding or performance of the current version of Multics, the long-range payoff in methodical engineering using better-understood memory structures is also evident.

Protection of programs and data

A long-standing objective of the public computer utility has been to provide facilities for the protection of executing programs from one another, so that users may with confidence place appropriate control on the release of their private information. In 1967, a mechanism was proposed[29] and implemented in software which generalized the usual supervisor-user protection relationship. This mechanism, named "rings of protection," provides user-written subsystems with the same protection from other users that the supervisor has, yet does not require that the user-written subsystem be incorporated into the supervisor. Recently, this approach was brought under intense review, with two results:

- A hardware architecture which implements the mechanism was proposed.[30] One of the chief features of the proposed architecture is that subroutine calls from one protection ring to another use exactly the same mechanisms as do subroutine calls among procedures within a protection area. The proposal appears sufficiently promising that it is included in the specifications for the next genera tion of hardware to be used for Multics.
- As an experiment in the feasibility of a multilayered supervisor, several supervisor procedures which required protection, but not all supervisor privileges, were moved into a ring of protection intermediate between the users and the main supervisor. The success of this experiment established that such layering is a practical way to reduce the quantity of supervisor code which must be given all privileges.

Both of these results are viewed as steps toward first, a more complete exploitation and understanding of rings of protection, and later, a less constrained organization of the type suggested by Evans and LeClerc[31] and by Lampson.[32] But more importantly, rings of protection appear applicable to any computer system using a segmented virtual memory. Two doctoral theses are under way in this area.

System programming language

Another technique of system engineering methodology being explored within the Multics project is that of higher level programming language for system implementation. The initial step in this direction (which proved to be a very big step) was the choice of the PL/I language for the implementation of Multics. By now, Multics offers an extensive case study in the viability of this strategy. Not only has the cost of using a higher level language been acceptable, but increased maintainability of the software has permitted more rapid evolution of the system in response to development ideas as well as user needs. Three specific aspects of this experience have now been completed:

- The transition from an early PL/I subset compiler[14] to a newer compiler which handles almost the entire language was completed. This transition was carried out with performance improvement in practically every module converted in spite of the larger language involved. The significance of the transition is the demonstration that it is not necessary to narrow one's sights to a "simple" subset language for system programming. If the language is thoroughly understood, even a language as complex as the full PL/I can be effectively used. As a result, the same language and compiler provided for users can also be used for system implementation, thereby minimizing maintenance, confusion, and specialization.
- Notwithstanding the observation just made, the time required to implement a full PL/I compiler is still too great for many situations in which the

compiler implementation cannot be started far enough in advance of system coding. For this reason, there is considerable interest in defining a smaller language which is easily compilable, yet retains the features most important for system implementation. On the basis of the experience of programming Multics in a subset of PL/I, such a language was defined but not implemented, since it was not needed.[33]

- A census of Multics system modules reveals how much of the system was actually coded in PL/I, and reasons for use of other languages. Roughly, of the 1500 system modules, about 250 were written in machine language. Most of the machine language modules represent data bases or small subroutines which execute a single privileged instruction. (No attempt was made to provide either a data base compiler or PL/I built-in functions for specialized hardware needs.) Significantly, only a half dozen areas (primarily in the traffic controller, the central page fault path, and interrupt handlers) which were originally written in PL/I have been recoded in machine language for reasons of squeezing out the utmost in performance. Several programs, originally in machine language, have been recoded in PL/I to increase their maintainability.

As with the earlier topics, the implications of this work with PL/I should be felt far beyond the Multics system. Most implementers, when faced with the economic uncertainties of a higher-level language, have chosen machine language for their central operating systems. The experience of PL/I in Multics when added to the expanding collection of experience elsewhere[34] should help reduce the uncertainty.

In a research project as large, long, and complex as Multics, any paper such as this must necessarily omit many equally significant ideas, and touch only a few which may happen to have wide current interest. It is the purpose of individual and detailed technical papers to explain these and other ideas more fully. The bibliography found in Reference 35 contains over twenty such technical papers.

Immediate future plans

The Multics software is continuing to evolve in response to user needs and improved understanding of its organization. In 1972 a new hardware base for Multics will be installed by the Information Processing Center at M.I.T. for use by the M.I.T. computing community. This program compatible hardware base contains small but significant architectural extensions to the current hardware. The circuit technology used will be that of the Honeywell 6080 computer. The substantial changes include:

(1) replacement of the high-performance paging drum initially with bulk core and, when available, LSI memory, and
(2) implementation of rings of protection as part of the paging and segmentation hardware.

Wherever possible the strategy of using off-the-shelf standard equipment rather than specially engineered units for Multics has been followed. This strategy is intended to simplify maintenance.

CONCLUSIONS

There are many conclusions which could possibly be drawn from the experience of the Multics project. Of these, we consider four to be major and worthy of note. First, we feel it is clear that it is possible to achieve the goals of a prototype computer utility. The current implementation of Multics provides a measure of the mechanisms required. Moreover, the specific implementation of the system, because it has been written in PL/I, forms a model for other system designers to draw upon when constructing similar systems.

Second, the question of whether or not the specific software features and mechanisms which were postulated for effective computer utility operation are desirable has now been tested with specific user experience. Although the specific mechanisms implemented subsequently may be superseded by better ones, it is certainly clear that the improvement of the user environment which was wanted has been achieved.

Third, systems of the computer utility class must evolve indefinitely since the cost of starting over is usually prohibitive and the many-year lead time required may be equally unacceptable. The requirement of evolvability places stringent demands on design, maintainability, and implementation techniques.

Fourth and finally, the very act of creating a system which solves many of the problems posed in 1965 has opened up many new directions of research and development. It would appear almost a certainty that increased user aspirations will continue to require intensive work in the areas of computer system principles and techniques.

In closing, perhaps we should take note that in the seven years since Multics was proposed, a great many other systems have also been proposed and constructed;

many of these have developed similar ideas.* In most cases, their designers have developed effective implementations which are directed to a different interpretation of the goals, or to a smaller set of goals than those required for the complete computer utility. This diversity is valuable, and probably necessary, to accomplish a thorough exploration of many individually complex ideas, and thereby to meet a future which holds increasing demand for systems which embrace the totality of computer utility requirements.

ACKNOWLEDGMENT

It is impossible to acknowledge accurately the contributions of all the individuals or even the several organizations which have given various forms of support to the development of Multics over the past seven years. As would be expected of any multi-organization project spanning several years there has been a turnover in the personnel involved. As the individual contributors now number in the hundreds, proper recognition cannot be given here. Instead, since the development of significant features and designs of Multics has occurred in specific areas and disciplines such as input/output, virtual memory design, languages, and resource multiplexing, a more accurate delineation of achievements should be made in specialized papers. So in the end we must defer to the authors of individual papers, past and future, to acknowledge the efforts of some of the many contributors who have made the evolution of Multics possible.

REFERENCES

1 F J CORBATÓ M M DAGGETT R C DALEY
An experimental time-sharing system
AFIPS Conf Proc 21 Spartan Books 1962 pp 335-344

2 P A CRISMAN Ed
The compatible time-sharing system: A programmer's guide
2nd ed MIT Press Cambridge Massachusetts 1965

3 F J CORBATÓ V A VYSSOTSKY
Introduction and overview of the Multics system
AFIPS Conf Proc 27 1965 FJCC Spartan Books Washington D C 1965 pp 185-196

4 E L GLASER et al
System design of a computer for time-sharing application
AFIPS Conf Proc 27 1965 FJCC Spartan Books Washington D C 1965 pp 197-202

5 V A VYSSOTSKY et al
Structure of the Multics supervisor
AFIPS Conf Proc 27 1965 FJCC Spartan Books Washington D C 1965 pp 203-212

6 R C DALEY P G NEUMANN
A general-purpose file system for secondary storage
AFIPS Conf Proc 27 1965 FJCC Spartan Books Washington D C 1965 pp 213-229

7 J F OSSANNA et al
Communication and input/output switching in a multiplex computing system
AFIPS Conf Proc 27 1965 FJCC Spartan Books Washington D C 1965 pp 231-241

8 E E DAVID JR R M FANO
Some thoughts about the social implications of accessible computing
AFIPS Conf Proc 27 1965 FJCC Spartan Books Washington D C 1965 pp 243-247

9 E I ORGANICK
The Mutlics system: An examination of its structure
MIT Press (in press) Cambridge Massachusetts and London England

10 R W WATSON
Timesharing system design concepts
McGraw-Hill Book Company New York 1970

11 W T COMFORT
A computing system design for user service
AFIPS Conf Proc 27 1965 FJCC Spartan Books Washington D C 1965 pp 619-626

12 A S LETT W L KONIGSFORD
TSS/360: A time-shared operating system
AFIPS Conf Proc 33 1968 FJCC Thompson Books pp 15-28

13 R E SCHWEMM
Experience gained in the development and use of TSS/360
AFIPS Conf Proc 40 1972 SJCC AFIPS Press (This volume)

14 F J CORBATÓ
PL/I as a tool for system programming
Datamation 15 6 May 1969 pp 68-76

15 R A FREIBURGHOUSE
The Multics PL/I compiler
AFIPS Conf Proc 35 1969 FJCC AFIPS Press 1969 pp 187-199

16 R J FEIERTAG E I ORGANICK
The Multics input-output system
ACM Third Symposium on Operating Systems Principles October 18-20 1971 pp 35-41

17 M T ALEXANDER
Organization and features of the Michigan terminal system
AFIPS Conf Proc 40 1972 SJCC AFIPS Press (This volume)

18 A BENSOUSSAN C T CLINGEN R C DALEY
The Multics virtual memory
ACM Second Symposium on Operating System Principles October 20-22 1969 Princeton University pp 30-42

19 *BASIC*
Fifth Edition Kiewit Computation Center Dartmouth College September 1970

* Some examples which have not already been mentioned include: the TENEX system of Bolt, Beranek and Newman, the VENUS system of Mitre Corp., the MU5 at Manchester University, RC-4000 of Regnecentralen, 5020 TSS of Hitachi Corp., DIPS-1 of Nippon Telephone, the Japanese National Computer Project, the PDP-10/50 TSS of Digital Equipment Corp., the BCC-500 of Berkeley Computer Corp., I.T.S. of the M.I.T. Artificial Intelligence Laboratory, Exec-8 of Univac, System 3 and 7 and the SPECTRA 70/46 of RCA, Star-100 of CDC, UTS of Xerox Data Systems, the 6700 system of Burroughs, and the Dartmouth Time-Sharing System.

20 *APL/360 user's manual*
IBM form number GH20-0683-1 March 1970

21 J H SALTZER
Traffic control in a multiplexed computer system
ScD Thesis MIT Department of Electrical Engineering 1966 Also available as Project MAC technical report TR-30

22 E W DIJKSTRA
The structure of the 'THE'-Multiprogramming system
Comm ACM 11 5 May 1968 pp 341-346

23 R W PARKER
The Sabre system
Datamation 11 9 September 1965 pp 49-52

24 R R SCHELL
Dynamic reconfiguration in a modular computer system
PhD Thesis MIT Department of Electrical Engineering 1971 Also available as Project MAC technical report TR-86

25 C J CONTI
Concepts for buffer storage
IEEE Computer Group News March 1969 pp 9-13

26 R A MEYER L H SEAWRIGHT
A virtual machine time-sharing system
IBM Systems Journal 9 3 1970 pp 199-218

27 F J CORBATÓ
A paging experiment with the Multics system
In Honor of P M Morse MIT Press Cambridge Massachusetts 1969 pp 217-228

28 M D SCHROEDER
Performance of the GE-645 associative memory while Multics is in operation
ACM Workshop on System Performance Evaluation April 1971 pp 227-245

29 R M GRAHAM
Protection in an information processing utility
Comm ACM 11 5 May 1968 pp 365-369

30 M D SCHROEDER J H SALTZER
A hardware architecture for implementing protection rings
ACM Third Symposium on Operating Systems Principles October 18-20 1971 pp 42-54

31 D C EVANS J Y LeCLERC
Address mapping and the control of access in an interactive computer
AFIPS Conf Proc 30 1967 SJCC Thompson Books 1967 pp 23-30

32 B W LAMPSON
An overview of the CAL time-sharing system
Computer Center University of California Berkeley September 5 1969

33 D D CLARK R M GRAHAM J H SALTZER M D SCHROEDER
Classroom information and computing service
MIT Project MAC Technical Report TR-80 January 11 1971

34 J E SAMMET
Brief survey of languages used for systems implementation
SIGPLAN Notices 6 9 October 1971 pp 1-19

35 *The multiplexed information and computing service: Programmers' manual*
MIT Project MAC Rev 10 1972 (Available from the MIT Information Processing Center)

APPENDIX: A CHECKLIST OF MULTICS FEATURES

Following is a checklist of currently available features and facilities of Multics. Although many of the features are described in cryptic and untranslated local jargon, one can at least obtain a feel for the range of facilities now provided. Further information on most of these features may be found in the Multics Programmers' Manual.[35]

Interactive Time-Sharing Facilities
- file editors
- file manipulation (rename/move/delete)
- personal command abbreviations
- recursive command language
- source language debugging with breakpoints
- subroutine call tracer
- can stop any running command or program

Programming Languages
- PL/I
- FORTRAN
- BASIC*
- APL
- LISP
- BCPL
- ALM (assembly language/Multics)

Information Storage System
- configuration independent
- accessed through virtual memory (segments)
- access control lists by user and project
- links to segments of other users
- hierarchical directory (catalog) arrangement
- public library facilities
- sharing at all levels
- multiple segment names (synonyms)
- separate control of read, write, and execute

Programming Environment
- segmented virtual memory
- dynamic linking of procedures and data, or prelinking
- interprocess communication
- independent of configuration
- uniform error handling mechanism
- user definable protection rings
- microsecond calendar clock with interrupt
- program interrupt signal from console

Input and Output
- standard typewriter interface for device independence
- ASCII character set used throughout
- input characters converted to canonical form
- erase and kill editing on typed input

- I/O streams switchable during execution
- magnetic tape, printer, card punch, card reader
- typewriter terminals: IBM 2741, 1050
  Teletype 37, 33, 35
  Dura, Datel, Execuport,
  Terminet-300
- graphic support library (devices: ARDS, IMLAC, DEC 338)
- ARPA network
- interfaces at three levels:
  - formatted data conversion
  - bit stream control
  - full device control

Management Facilities
- passwords required for login
- project may interpose authentication procedure
- decentralized projects
- accounting, billing, and quotas
- on-line probing and account adjustment
- operator or system initiated logout of users
- unlisted and anonymous users
- limited service system
- dynamic reconfiguration of memories and processors
- system performance metering for parameter adjustment
- project-imposed starting procedure

Communication Facilities
- interuser mail
- help command; help files
- message of the day
- on-line error reporting and consultation service
- on-line user graffiti board
- operations message broadcast to logged-in users

Absentee Facilities
- priority/defer queues for printer, card punch
- queued translator facility
- general absentee job facility

Reliability Measures
- weekly file copies onto tape
- daily disk/drum copy onto tape
- incremental file copies onto tape, ½ hour behind use
- salvager to clean up files after system crash
- emergency shutdown entry to system

Maintenance Features
- on-line library change, no disruption of current users
- entire system source on-line, maintenance tools
- system checkout on small hardware configuration
- on-line performance monitoring of
  - multiprogramming
  - paging traffic
  - drum/disk usage
  - typewriter traffic
- user performance feedback:
  - cpu time and paging load on each command
  - page trace always operating
  - subroutine call counters

Private Project Subsystems
- project providable command interface
- Dartmouth environment*
- student environment

Miscellaneous Facilities
- desk calculators
- sort command
- memorandum formatting and typing subsystem
- user-provided list of programs to be automatically executed when user logs in
- GCOS environment

* The BASIC system and the Dartmouth environment were developed at Dartmouth College. They are used at M.I.T. by permission of Dartmouth College.

Organization and features of the Michigan terminal system

by MICHAEL T. ALEXANDER

University of Michigan
Ann Arbor, Michigan

INTRODUCTION

This paper will explore some aspects of the Michigan Terminal System (MTS) developed at the University of Michigan. MTS is the operating system used on the IBM 360/67 at the University of Michigan Computing Center, as well as at several other installations. It supports a large variety of uses ranging from very small student-type jobs to large jobs requiring several million bytes of storage and hours of processor time. Currently at the University of Michigan there are about 13,000 users running as many as 86,000 jobs per month.

MTS was developed almost entirely by the staff of the University of Michigan Computing Center, although some components were developed by other installations using it, and many compilers and subsystems were borrowed from other systems. Fewer man-years have been spent developing and maintaining MTS than most comparable systems (about 50 man-years total for both development and maintenance), and some of the organizational aspects that allow this will be explored here. A few of the more unusual aspects of the system will be considered, along with an attempt to evaluate their effectiveness.

HISTORY

It will be useful to consider the history of the development of MTS in order to provide the proper perspective for later discussions. The course of its development was rather unusual and influenced some of the features to be mentioned later.

In 1965 the University of Michigan Computing Center made plans to install an IBM 360/67[4] in late 1966. It was planned to use the standard IBM operating system (Time-Sharing System/360[6]) for this machine rather than to develop an operating system for it. Since the model 67 includes special features to simplify time-sharing, a special operating system was required for it. The Computing Center acquired an IBM 360/50 for use both in performing peripheral operations for the IBM 7090 then in use and in checking out 360/67 programs to be used with TSS. In May 1966 development of a very small operating system for this machine was initiated by two members of the Computing Center staff. This system had very modest goals: to allow one or two Computing Center staff members to run certain 360 programs, and to handle at the same time the peripheral operations for the 7090. These goals were reached within a few months, and the system was in use by Computing Center staff by the first part of 1966.

When the 360/67 was delivered in early 1967 the plan was still to run TSS as soon as it reached an acceptable level of reliability. In the meantime, the 360/50 software system was moved over to the 360/67, but it still did not use any of the special hardware on the 360/67.

At this point MTS was a rather primitive system which would allow the simultaneous use of the machine by one or two batch jobs and three or four conversational users. The facilities available included most of the essentials but very few frills, and the system was used almost exclusively by Computing Center staff members. However, from the very beginning the system had been designed to be open ended, so that more facilities could be easily added.

By the spring of 1967, it was decided to make MTS available to a limited number of persons not on the Computing Center staff who were most interested in conversational computing. This required a number of changes, the most important of which was the addition of accounting procedures to charge for machine usage, and the decision also provided an impetus to further augment the system. Although the system was far from complete at this time, it had already been in daily use for several months.

The first of several major changes in the system

TABLE I—Summary of MTS Usage for One Year

	Number Jobs	CPU Time (Hours)	Lines Printed	Pages Printed	Cards Punched	Cards Read
DEC. 1970	68040	400	28520413	835019	387423	6785972
JAN. 1971	48909	304	20564780	597105	274767	4050356
FEB. 1971	64734	388	25029078	740910	259886	5323226
MAR. 1971	81894	481	32936379	982605	318247	7623300
APR. 1971	57558	356	24968176	754722	455143	6100019
MAY 1971	35141	232	16487547	449463	216799	3741426
JUNE 1971	62079	431	28782123	804884	454602	6070196
JULY 1971	57927	415	27112847	765186	422543	4970674
AUG. 1971	55996	379	25756018	734292	453018	4743966
SEP. 1971	56302	301	20030212	567504	325344	3659216
OCT. 1971	86107	380	27795205	830841	404359	5773416
NOV. 1971	90243	394	29264425	879604	381753	6268137

occured in November 1967 when it was modified to use both the special relocation hardware of the 360/67 and the paging drum. This required replacing one component and adding two components to the system, and was done without any changes to user programs or procedures. In fact it was so transparent to users of the system that for several weeks after it was put into use people were still asking when it would be installed. More importantly, the change was transparent to the rest of the system, most of which was not even reassembled. The most notable external sign of the change was a sudden eightfold jump in system capacity.

Other changes of similar magnitude have been made since then with equal success. For example, the supervisor was changed in September 1968 to support hardware configurations with more than one central processing unit. This change was made with modifications to only two components. Another example was the addition of remote job entry to the batch monitor, which was done without significant changes to other modules.

The resulting system is capable of serving the needs of a large and sophisticated user population. On a typical day there will be about 70 people using the system from terminals with any remaining processor time being used by up to 5 or 6 batch jobs. This load runs with response times of about a second on a two processor machine with 1.5 million bytes of storage. Table I gives some information concerning the usage of the system in the last year and Table II gives more details concerning November 1971.

In summary, MTS has grown from a very small, simple system to a sophisticated one that supports a wide variety of applications for over 20,000 users at five installations. This growth has been achieved without requiring reprogramming or control language changes by users of the system. This evolutionary approach to system development contrasts with the more usual approach in which most of the system is designed on paper before any of it is in regular use. The evolutionary approach has a number of advantages and perhaps should be more frequently considered. It allows the designers of the system to use it and become aware of its advantages and disadvantages while it is still possible to change parts of the design with relative ease. It also allows the best people to work on the whole system since they can move from component to component as the system develops. On the other hand, it requires that the designers be able to foresee possible

TABLE II—MTS usage during November, 1971

TOTAL NUMBER OF JOBS RUN	90243
TOTAL NUMBER OF BATCH JOBS RUN	52903
TOTAL NUMBER OF TERMINAL SESSIONS	37340

TOTAL CONNECT TIME: 10735 HOURS
TOTAL CPU TIME: 394 HOURS
MAXIMUM NUMBER OF TERMINAL USERS SIGNED ON AT ONE TIME: 72

		Total	Batch	Terminal
Number of Jobs Run per Day,	AVERAGE:	3008	1763	1245
	MAXIMUM:	4825	3079	2076
	MINIMUM:	171	94	44
Number of Jobs Run 8 TO 12,	AVERAGE:	2790	1606	1185
	MAXIMUM:	4363	2691	1901
	MINIMUM:	9	1	8
Number of Jobs Run 9 TO 5,	AVERAGE:	1492	718	774
	MAXIMUM:	2311	1311	1215
	MINIMUM:	0	0	0
Number of Jobs Run per Hour,	AVERAGE:	125	73	52
	MAXIMUM:	377	295	205
	MINIMUM:	0	0	0

directions for change and to allow for them when designing parts of the system.

ORGANIZATION AND IMPLEMENTATION OF MTS

In order to be able to extend and modify the system easily, it was organized as a set of independent components with well-defined interfaces between them. This idea is, of course, neither new nor unique; but there are differences in this respect between MTS and other systems. For one thing the term "component" as used here refers to a rather large fragment of the system, for example the entire supervisor. Also the interfaces between components are much more rigid than in many systems. The result is that components are independent of one another to the extent that it is quite easy to replace almost any one of them without affecting any other. Quite major changes can be introduced this way with relatively little work or trouble.

The dependence of components on one another is reduced by the fact that most components communicate with only a few others. Almost all components communicate with the supervisor, and with at most one or two other components. The interface with the supervisor is the same for all components and is the most important in the system. Dependency is further reduced by allowing very few special cases in interfaces; for example, all input/output operations are done using the same supervisor facilities whether the input/output is for a card reader, a paging drum, or anything else.

It is very important that any tables or control blocks maintained by the system should be directly referenced by as few programs as possible. For example, the control blocks relating to the file system should be accessed only by the subroutines comprising the file system. Other programs that wish to access a table or control block should call a subroutine that is part of the component that "owns" it. It is not enough to simply parameterize the format of the table or control block, since even finding all components that refer to it and reassembling them can be a very difficult and time consuming project. MTS is superior in this respect, and this is one major reason the system has developed smoothly. It is often necessary to reformat some system control block and it must be possible to do so easily.

One interface that must be well-designed is the one between the system and the application program. This must be the cleanest of all since there is no hope of changing the application programs when the system changes. In particular, no user program should ever refer directly to any system control block. If the information is needed it should be made available through a subroutine in the system. If the hardware allows it at a reasonable price, all system control blocks should be protected from user programs for both reference and change.

Since most large programs written for the IBM/360 are written to run under IBM's OS/360 system, it was desirable to make it easy to convert programs written for OS/360 to run in MTS. For this reason MTS provides many of the same services as OS/360. It also provides many other services and most of the similar ones are different in detail, requiring some modification to OS/360 programs. However, programs written in Fortran or PL/I can normally be run without modification since all dependency on OS/360 is in the run time library which has been converted to run in MTS. Furthermore, there is a program that allows some OS/360 object programs, such as the COBOL compiler and its object programs, to run in MTS by simulating exactly the services that are provided by OS/360.

With large components and well-defined interfaces it has been possible to divide the programming effort for MTS vertically rather than horizontally. By this it is meant that one or two individuals are assigned responsibility for a component and then follow it from design through implementation and even maintenance. The person responsible for a component is given considerable freedom to design the internal structure of the component so long as the interfaces are maintained unchanged. The fact that he also must implement and maintain the component means both that the design will be realistic and that it will be conformed to. This organization avoids part of the trouble many systems have had in trying to translate the design into code, since the people writing the code are not novices and thoroughly understand the design of the component.

Most components of the system can be debugged and many can be installed while the system is running without affecting the reliability of operation. It is possible to substitute a private copy of all components except the supervisor, the batch monitor, parts of the command language, and certain other similar things. It is also possible to replace a module without shutting down in most cases. Those modules that cannot be debugged in this way can be debugged by running the system in a "virtual machine" under the production system. The virtual machine program allows any system designed to run on a 360/67, including OS/360, MTS, or any other "stand alone" program, to be run. This type of operation does not greatly affect reliability. Indeed the software reliability of MTS is good. Between September 1970 and September 1971 there were 38 software caused system crashes. This is only 9 percent of the total crashes in this period, 85 percent of which were caused by hardware failure. During this time many parts of the system were debugged online.

FILE AND DEVICE MANAGEMENT

Perhaps the most interesting technical innovation in MTS is found in the methods used to support different types of input/output devices and files. These methods also provide a good example of how a well-defined interface can solve many problems.

A major problem in the implementation of any operating system is providing software support for the many different types of input/output devices to be used with it. This can consume a very large amount of time and energy. Furthermore, if the system does not provide truly device-independent input/output it will be much more difficult to debug and run programs under it. The approach taken toward I/O in MTS solves these problems better than in most commonly used systems.

The first and most important consideration concerning the I/O structure in MTS is that all input/output, whether by MTS itself or by a program running under MTS, is done by the same set of subroutine calls. These calls are independent of what program is doing the I/O and of the type of file or device being used. This means that any program including any part of MTS can read or write from any type of file or device with no code changes. This extends to all devices including typewriter and graphic terminals, unit record devices, files, magnetic and paper tape, etc. Of course a program may be designed to take advantage of special characteristics of a particular device type—for example, it might use the vector generating capabilities of a graphics terminal—and if it does so it loses some of its device independence.

Any call to the input/output routines in MTS specifies, either implicitly or explicitly, a quantity known as a File or Device name (FDname). The call may specify a Logical Unit Name to which an FDname has been attached, or it may specify the internal representation of a specific FDname returned by a subroutine for the purpose. An understanding of the concept of an FDname is crucial to an understanding of MTS input/output processing.

The simplest form of an FDname is simply the name of some file or device known to the system. For example,

SOURCEFILE

An FDname can also specify a subset of a file by appending a line number range to the simple name:

SOURCEFILE (10,20)

specifies lines 10 though 20 of the file SOURCEFILE. It is also possible to specify "modifiers" that determine special processing to be done for I/O on this FDname. For example, to convert all data read or written to upper case one could use

SOURCEFILE@UC

This will not affect the contents of the file on read operations, but only the data read or written using this FDname. More complicated FDnames can be constructed by concatenating the simple ones described so far. For example,

SOURCEFILE1(10,20)+SOURCEFILE2(−23,32)
+(100.75,123)

specifies lines 10 through 20 from SOURCEFILE1 followed by lines −23 through 32 and lines 100.75 through 123 from SOURCEFILE2. This type of concatenation is called explicit concatenation; it is also possible to specify concatenation implicitly by the data read. If a line read from any FDname is of the form "$CONTINUE WITH FDname [RETURN]" then the contents of the FDname given in this line will be used for further input, followed by the remaining lines from the current FDname if RETURN is specified. This action can of course be turned off to allow such records to be read directly. This generality in specification of FDnames is the single most useful feature of MTS and allows great flexibility in the use of commands and files.

It is difficult to design a set of input/output subroutines that will be suitable for all different types of input/output devices, and inevitably certain compromises must be made. It was decided to treat input/output as a transmission of a series of logical records, not as an unbroken stream of characters. It seemed that too many input/output devices had "natural" record boundaries that are hard to ignore. Hence the basic subroutines are subroutines to read or write a single record to or from an input/output device or file. In general the approach has been to make the various different files and devices look as much as possible like line files, which consist of a sequence of records associated with line numbers.

It was intended to develop an input/output system designed so that it would be easy to add support for any type of device that would fit into the structure described above. It was necessary that it be possible to debug support routines, and to install new ones, without disrupting normal operation of the system. It was also necessary that all code peculiar to a particular type of device be located in a single component of the system. To meet these and other goals the concept of a Device Support Routine (DSR) was created. There is a DSR for each type of device supported by MTS (some 20 in all) and each one supports one class of devices, for example low speed telephone lines, line files, magnetic tape drives, or some similar category of file or device.

The interface between a DSR and MTS is completely independent of the type of file or device. Every DSR has the same eight entry points and when MTS calls one of them it is completely unaware of the nature of the device. When a DSR is called at one of its entries it is responsible for interpreting the parameters properly for the device in question and for initiating, via calls on the supervisor, any actual input/output operations. The communication between MTS and the DSR is on a record level, while the communication between the DSR and the supervisor or the device is on a channel program level.

A DSR may be either private to a single task or shared among some or all tasks using the appropriate type of device. Furthermore any task may use a private copy of a shared DSR, for example to allow a DSR to be debugged. Using this technique it was possible to completely rewrite and install the DSR for magnetic tapes without any time on a dedicated machine; all development was done under regular production MTS without any adverse effects on the rest of the system. This was possible largely because the interface between MTS and the DSR is extremely clean and each side is independent of the other. This interface is described in reference 8 on pages 353 and following.

VIRTUAL MEMORY AND MULTI-PROCESSING

Two of the aspects of the system that attract the most attention are its use of virtual memory[2] with demand paging and the fact that it supports multiple central processors. The most interesting thing about these is that both were added to the system after it was already operational. This provided a good opportunity to observe both the implementation difficulty and the effect on performance of paging and multi-processor support.

Adding virtual memory to the system required about six man-months of work and improved performance at least tenfold. The performance comparison should not be taken too seriously since the pre-paging version of MTS did not make use of any swapping techniques, but rather kept all programs in main storage at all times they were in use. A more meaningful comparison is currently being attempted at the University of Newcastle upon Tyne in England where a version of MTS that runs on a standard 360 and uses swapping has been prepared. It is planned to perform some comparisons between this system and the model 67 version of MTS.

Modifying the system to support multiple processors required only about four man-months of work. About half of this was spent writing code to handle the more complex input/output hardware configurations possible with larger versions of the model 67, and the rest was spent adding code to the supervisor to avoid simultaneous destructive access to data by more than one processor. To do this required a great deal less work than most people would think, with quite effective results.

Measurements have been made of the amount of degradation due to interference between processors in a two processor configuration. There are two major sources of interference in such a machine: one processor may have to wait to gain access to data the other is modifying, and the increased load on main storage may cause both processors to run slower. The first source of interference was measured directly in MTS and turned out to result in a performance degradation of 1.5 percent when the system was operating under a very heavy load. It is interesting to note that the average time that a processor had to wait for access to data was only 7.6 microseconds. The second source of interference was measured indirectly by running the same job stream on different configurations and results indicate that a degradation of about 3.6 percent occurs for this job stream and that it is highly variable.

FILE SYSTEM

In any modern operating system the file system that allows users to store and retrieve data in the system is most important. This is even more true of systems such as MTS that encourage conversational use. Unfortunately this is not one of the strong points of MTS and perhaps it would be instructive to look at some of the advantages and disadvantages of the MTS file system.

The MTS file system supports two types of files. "Line files" consist of a set of lines, or records, each of which is associated with a line number which is not part of the record. Lines may be read, written, inserted, or deleted by line number or sequentially. Because of the implementation no line may be longer than 255 characters and no file may be larger than about 300,000 characters. The ability to write by line number without knowing whether the line already exists in the file, and the ability to read and write sequentially, ignoring the line numbers, make line files very useful and easy to use, but the restrictions noted above are severe limitations.

The second type of file, "sequential files," consist of a set of lines or records which normally are read or written sequentially, and which do not have line numbers. This type of file was invented later both because of the above restrictions and in order to reduce the number of accesses to the direct access devices, particularly the IBM 2321 Data Cell Drives. If line files had been better designed, sequential files would not have been necessary.

The file system in MTS gives the user no opportunity or need to worry about the actual format of the information on the direct access devices. This means that the use of these devices can be optimized by using a good format and efficient input/output techniques. On the other hand, very sophisticated users with specialized file requirements could in some cases improve performance by specifying some of this information if it were possible. However, very few people have the inclination or ability to do this; more importantly, a good operating system should not require them to do so to get good performance. Furthermore, any single user may degrade total system performance by optimizing his own performance, for example by using too much channel time and thereby locking other users out of their files.

In the opinion of the implementors of MTS one major disadvantage of the file system is that it is independent of the paging system. It would be much better to treat file I/O in the same way as paging I/O, as is done in Multics.[1] This would allow much better optimization of direct access device operations and it would allow users to treat files as extensions of their virtual memory. It is hoped that someday such a virtual memory file system may be installed in MTS.

Another major disadvantage that we see in the MTS file system is that it uses a direct access volume format compatible with that of OS/360.[5] This format is very restrictive and slow with respect to allocation and deallocation of space. It would be much better to use a format similar to that of TSS/360,[7] in which the concept of an "extent" of contiguous space does not exist.

BATCH VERSUS TERMINAL USAGE

When development of MTS was started it was anticipated that it would be primarily a terminal system with batch or nonconversational usage playing a much more limited role. This prediction turned out to be completely wrong. In fact, the use of terminals started out on a much smaller scale than the use of batch, and only recently have the two modes become about equally popular. There are probably several reasons for this, not the least of which is habit. From the beginning MTS has made batch and terminal usage as nearly compatible as possible and equally convenient, and many people simply continued to use the computer in the way they were used to. Other reasons are that in the early days there were relatively few terminals available and they were rather expensive to use. Both of these problems have been largely corrected, but those who were burned are reluctant to try again.

It is important that a system provide compatible batch and terminal modes of operation with the same command language and facilities. Most uses of the system will require both modes of operation to some extent and switching back and forth is difficult if the modes are not compatible. In particular it must be possible to run any program in either mode.

FUTURE TIME-SHARING SYSTEMS

It is difficult, even foolhardy, to make specific predictions for the future in computing, but some trends seem clear at this time. For example, there are more and more terminals available and they are becoming more and more sophisticated.

There is a current trend toward computer networking which is approaching the point where practical networks will soon be operational. Most work in this area so far has focused on the technical problems of connecting dissimilar isolated systems, while the difficult problem of what is to be transmitted has been largely ignored. When networks become truly operational, there will be very difficult problems with standards for data formats, data description languages, programming languages, etc. These problem of course already exist, but the closer connection between systems resulting from networking will greatly aggravate them.

Another problem which must be overcome before networking is really practical is the administrative problem. When networks include nodes that must bill for all services rendered it becomes difficult to integrate these nodes into the network unless some convenient way can be found to bill for services provided to other nodes in the network. This may seem like a trivial problem; but since a very large number of end users may be involved, making it difficult for each node to bill them directly, and since nodes in the network will vary widely in accounting standards and practices, it is not trivial at all.

The term "computer utility" is used so frequently these days that is has almost lost any meaning. It does not accurately describe any existing system although some systems such as MTS include some aspects of it. However, we are approaching the time when we will see the first true computer utilities, and it is worthwhile to look at some of the problems that may be encountered.

One thing that seems obvious at this point is that most customers of the computer utility of the future will have little programming knowledge. Most programming will be done by organizations and individuals that wish to sell a problem-solving service to people who know little or nothing about programming. Some of these services will be provided by those who provide the computer; many or most, however, will be provided by third parties.

The computer utility of the future must somehow separate the customer from many of the details of the computing system he is using. There is no reason the customer needs to know, or should know, about such things as the characteristics of the direct access devices he is using. To involve the end user in this sort of thing is both a waste of his time and unduly restrictive to those operating the computer.

Closely related is the same old problem of compatibility and standardization. The users of the computer must be able to switch from one system to another with a minimum of necessary changes. Those who are developing programs for use by nonprogrammers will not be able or willing to rewrite them for the peculiarities of every computing system. To do so would be like having every TV station using its own video transmission techniques.

To allow problem solving services to be sold by third parties will require facilities which are not available in most current systems. The programs and data which comprise such a service will be stored in files in the system which must of course be protected from unauthorized access. In addition there must be facilities to allow for proper accounting by those providing the service. Implementation of this will require much better protection of different levels of code within the system. The main limitation to this presently is the lack of suitable hardware on most machines available today. The IBM 360 series, including the model 67, is particularly bad in this respect, since it does nothing to help implement levels of protection within a task as is done on the Honeywell 645.[3]

Better protection of this sort is also required in order to be able to solve some of the problems that are becoming acute concerning privacy and data bases. These problems, however, require more than technical innovation to solve. In general they are problems that have been around for a long time, but which had not been acute when the harmful information was so widely distributed as to be almost inaccessible. The role of the computer industry in the solution of these problems would seem to be to provide the technical tools and advice that will allow those outside the industry who must find solutions to do so.

There seems to be a trend away from the type of multiprocessor organization found in the 360/67 in which all processors and input/output equipment is connected symmetrically, and this is disturbing. The symmetric type of organization found in the 360/67 is much easier to support in software since no part of the system need ever be aware of which processor it is executing on, as is necessary in machines such as 360 M65MP in which certain input/output equipment is accessible from only one processor. Furthermore the 360/67 organization is very flexible in that almost any single hardware component can be removed from the system and the system can still run. This has been very valuable at the University of Michigan, where this capability has resulted in about 10 percent more "up time" for the system as a whole.

On the other hand, there seems to be growing acceptance of address translation hardware such as appears on the 360/67. This hardware is very valuable for a variety of reasons and should be on all machines in the future. It has rarely been exploited to its fullest but even in present systems it more than pays for itself.

In summary, it is clear that computing is still in its adolescence and that we are just beginning to involve the nonprofessional in it. This change will be both exciting and painful as we face the technical and ethical problems of the future.

REFERENCES

1 R C DALEY P G NEUMANN
A general-purpose file system for secondary storage
AFIPS Conf Proc 27 1965 FJCC pp 213-229

2 P J DENNING
Virtual memory
Computing Surveys 2 Sept 1970 pp 153-190

3 R M GRAHAM
Protection in an information processing utility
Comm ACM 11 May 1968 pp 365-369

4 *IBM system 360 model 67 functional characteristics*
Form GA27-2719 IBM Corp NY

5 *IBM system 360 operating system data management services*
Form GC26-3746 IBM Corp NY

6 *IBM system 360 time sharing system concepts and facilities*
Form GC28-2003 IBM Corp NY

7 *IBM system 360 time sharing system—System data management facilities*
Form GC28-2056 IBM Corp NY

8 *MTS: Michigan Terminal System*
Vol I and II Second Edition University of Michigan Computing Center

9 *MTS: Michigan Terminal System*
Vol 1-14 Third Edition University of Michigan Computing Center

Regulatory developments in data communications—The past five years

by PHILIP M. WALKER

Georgetown University Law Center
Washington, D.C.

INTRODUCTION

Since its creation in 1934, the Federal Communications Commission has exercised regulatory jurisdiction over all interstate common carrier communications service in the U.S. The Commission has broad powers to review rates and tariffs, determine investment and operating practices, approve new plant installations, and literally shape the entire communications carrier industry, but historically the Commission has failed to exercise these powers vigorously. Instead it, like most government regulatory agencies, has tended to become dominated by the industry it supposedly regulates, and has come to place the status quo high on its list of values to be protected; until recently, that is.

Communications carrier services in the U.S. are today provided on a monopoly, or at best, duopoly basis. Each telephone company has a monopoly in providing dial telephone service in its geographic operating area, and although Western Union provides private line services "competitive" with those of the telephone companies, WU's service offerings and rates are identical to those established by the Bell System and copied by all independent (non-Bell) telephone companies as well. Bell's mammoth size (serving over 83 percent of all U.S. telephones) as compared with WU and the independent telephone companies has made the terms "Bell" and "communications carrier industry" virtually synonymous, with effective competition within the industry nonexistent. Thus, decisions regarding introduction of new services and equipment, rates, and tariff regulations have been made by a single firm, with minimal federal and state regulatory control in most cases. The result with respect to data communications has been a sluggish response to user demand for new and improved services (especially felt by data users during the early and mid-60s), restrictive tariffs which impeded the user's ability to obtain maximum benefit from the communications network, and higher costs than might prevail in a competitive atmosphere.

Fortunately for the communications user, the past five years have seen tremendous changes in this picture. Some of these changes are the result of the rapid growth of computing and data communications, and the carrier's natural response to that growth, but the most fundamental changes are in the regulatory and policy environment. After decades of apparent satisfaction with regulated monopoly as the *modus operandi* of the communications industry, the FCC seems to have done a turn-about and embraced competition as a means of stimulating innovation and efficiency in this industry. For example, in its landmark *Carterfone* decision in 1968,[15] the Commission broke the carriers' monopoly on the supply of terminal equipment and systems to be attached to or interconnected with the telephone network. The following year, in its equally precedent-setting *MCI* decision,[8] the Commission authorized the first specialized private-line carrier to enter the market and compete with Bell and Western Union. A year later the FCC, urged by the White House,[16] reversed its previous attitude toward the establishment of domestic communications satellite systems and invited applications for competitive systems from all interested parties.[6]

More recently the Commission has somewhat relaxed its attitude toward the competitive offering of hybrid data processing-message switching services,[12] and has opened the way for cable television (CATV) systems to enter the major metropolitan markets and supply a wide variety of new communications services.[7] During this same five-year time period, President Johnson's Task Force on Communications Policy issued a comprehensive report[27] endorsing the wider use of competition in the communications carrier industry, and several years later President Nixon's new Office of Telecommunications Policy also began to advocate increased reliance upon the forces of competition to achieve our national communications goals. This paper attempts to provide an overview of these developments, and to assess their importance for the future.

THE FCC COMPUTER INQUIRY

In late 1966, the Federal Communications Commission began what has since become a widely-discussed public Inquiry into the "Regulatory and Policy Problems Presented by the Interdependence of Computer and Communication Services and Facilities."[9,21] Rapid developments in digital technology, and corresponding growth of the computer equipment manufacturing and computer services industry had been on what some have termed a collision course with telecommunications and the communications common carriers. Several controversies in recent years brought this home to the FCC and pointed out the need for policy guidance.[17,18] The FCC's reaction was to publish a list of broad and interrelated questions and to invite comments from all interested parties, hoping thereby to develop the framework for such policy formulation. Over sixty responses to this Notice of Inquiry, totaling over 3,000 pages, were received by the Commission before its deadline in March, 1968.

To assist in evaluation of this material, the Commission then engaged Stanford Research Institute, which submitted its report in early 1969—summarizing the responses and analyzing the issues, but raising more questions than it answered.[29] Soon thereafter, the Commission issued a Report and Further Notice of Inquiry,[10] asking for comments on the SRI report. A year later, after receiving additional comments from some of the parties to the Inquiry, the Commission issued its Tentative Decision.[11] Finally, after receiving another round of written comments from, and hearing oral argument by, the parties, the Commission in March, 1971 issued its Final Decision and Order.[12] As might be expected during the course of a five-year proceeding, some of the issues originally addressed in the Inquiry were overtaken by the course of events and became moot by the time of the Commission's final decision; unfortunately, certain other issues were not adequately resolved in the final decision, and at the time of this writing they remain a source of uncertainty.

Regulation of computer services

Let us now consider the issues addressed by the FCC in this Inquiry. First was the question of whether, and under what circumstances, computer-based data processing or information services—remote-access services employing communication lines, and also perhaps stand-alone services—should be deemed subject to FCC regulation under the Communications Act of 1934.

The anticipated growth of the "computer utility" concept, much-discussed during the mid-60s, had suggested to some observers the possible need for some sort of government regulation of that utility. On many issues in the FCC Inquiry there developed a dichotomy of opinion, with the communications common carriers on one side and the computer firms and users on the other; but regarding this first question there was general agreement that no need for government regulation of computer services exists or is likely to exist in the future.

There are a number of factors which may indicate the need for regulation of an industry, such as the existence of natural monopoly characteristics which make effective competition impossible in the industry (e.g., the electric utility industry), or compelling requirements that the quality and integrity of the industry's product be insured (e.g., the pharmaceutical and banking industries). Comprehensive rate and profit regulation is imposed upon the common carrier communications industry because of its natural monopoly characteristics (continually increasing economies to scale) and the public interest requirement that the service be offered on a regular, well neigh universal basis. The data services industry exhibits none of these qualities; on the contrary, it is thriving in an atmosphere approaching the pure competitive model. Concern for the protection of the consuming public has led to certain types of regulatory controls in other industries such as insurance and banking. On this score, also, data and information services do not appear to warrant regulatory attention. Significant abuses have not arisen yet, and there is no reason to believe that they will occur in the future. The FCC therefore concluded that "In view of all the foregoing evidence of an effective competitive situation, we see no need to assert regulatory authority over data processing services whether or not such services employ communications facilities in order to link the terminals of subscribers to centralized computers."[11]

The protection of privacy

A second question asked by the FCC concerned the need for legislative or other government action to protect the privacy and security of data stored in computers that are interconnected with common carrier communication lines. The respondents discussed a number of protection measures available to users and system operators, and generally felt that FCC intervention or regulatory control was not needed at this time.

The Commission, noting that the privacy and security issues extended well beyond the scope of its jurisdiction to regulate communications, and observing other efforts taking place in Congress and elsewhere in government concerning this problem, elected to follow the advice of the respondents and refrain from taking any action pending further developments or abuses. While this was probably the best course for the Commission to follow for the present, it seems clear that the multitude of privacy and security problems associated with widespread usage of the computer, and especially the large multiple-access system, in all areas of our society is growing as fast as—or perhaps faster than—the use of such systems.[20,22] Government controls of some sort thus appear inevitable, and the FCC will probably play a role in future years in attacking that part of the problem which falls within its regulatory jurisdiction. Proposals have been made for the creation of a new federal agency to provide regulatory control of the computer-privacy area,[22] and such an approach, if adopted, might reduce or eliminate the FCC's role in this area. But it seems at least equally likely that Congress may choose instead to provide for the protection of computer data-bank privacy and security through expansion of the regulatory jurisdiction of the FCC and other relevant federal agencies rather than through creation of a new agency.

Adequacy of facilities for data transmission

The third topic in the Inquiry which the FCC disposed of without taking any action concerned the adequacy of common carrier services and facilities for data transmission. Many of the computer-industry firms and users who participated in the Inquiry identified serious deficiencies in the service offerings of the telephone companies and Western Union, circa 1968, and urged certain modifications and improvements.[21] These included the provision of additional channel bandwidths, the development of a widespread switched broadband service, a reduction in the three-minute minimum charging period on the dial network, a reduction of error rates in data transmission generally, more rapid introduction of digital transmission systems such as T-carrier and digital microwave, and lower costs.

The FCC recognized these needs of the computer community, but concluded that several other events which had occurred during the pendency of the Inquiry obviated the need for any Commission action in the Inquiry docket. These external events included the FCC's approval in 1969 of the first specialized carrier, MCI, to compete with the established carriers[8,21,30,31] (discussed below); the Commission's landmark *Carterfone* decision in 1968 which permitted competitive supply of subscriber terminal equipment for use on the telephone network[15,21] (discussed below); and the advent of a number of new and improved data transmission services being planned and phased in by the existing carriers—partly in response to the new competition created by *MCI* and *Carterfone*.[30] In effect, the Commission opted to rely upon this new competition to achieve its goals of improved common carrier service for data transmission, rather than taking direct action itself to attempt to improve the carriers' performance.

While these competitive developments certainly have had a beneficial effect upon the quality and adequacy of the carriers' data transmission service, it is unfortunately not clear that all of the user's problems have been solved—or that they will be in the future. The telephone industry, as with most regulated quasi-monopoly public utilities (for the vast majority of the telephone carriers' revenues are safe from competitive inroads), continues to respond to customer demand rather sluggishly when measured by a competitive-industry yardstick. Telephone company performance is on the uptrend, but so are rates in many cases, and careful regulatory agency attention will be needed in the coming years—at both the federal and state levels—to insure that the user obtains maximum value for his data communications dollar. In the author's view, it is unfortunate that the FCC chose to sweep this problem under the rug in its final order in the computer inquiry.

Common carrier tariff restrictions

One additional issue in the Inquiry, very important when it was posed in 1966, was later found by the FCC to have been also largely mooted by ensuing events. This concerned the reasonableness of the carriers' traditional tariff prohibitions of (1) customer-owned "foreign" attachments on the telephone network, (2) interconnection to the network of customer-owned communications systems, and (3) customer sharing or resale of carrier lines. The Commission's 1968 *Carterfone* decision forbade unreasonable carrier restrictions upon the user's freedom to attach or interconnect harmless terminal equipment or communications systems, and the carriers subsequently made certain changes in their tariffs to comply with this ruling.[21] As discussed below, considerable disagreement remained as to whether the carriers went far enough in compliance with *Carterfone*, but the Commission had instituted procedures for continued discussion and study of this question, and felt that no additional action was required in the computer inquiry itself.

Concerning sharing and resale of private line channels released from the carriers, several things occurred during the pendency of the computer inquiry. First, MCI had previously proposed to offer sharing in its new private line service pending before the Commission. Sensing the likelihood of FCC approval of MCI's license applications (which in fact occurred in August, 1969), and perhaps also influenced by the numerous users' requests for sharing voiced in the computer inquiry, the Bell System in February, 1969 relaxed its tariffs to permit sharing ("joint use") of its voice-grade and telegraph-grade private lines.[21]

This relaxation of the prohibition against sharing did not apply to channels of greater than voice bandwidth, for most broadband channels are leased under the Telpak tariff at greatly reduced bulk rates, and sharing of such channels raises complex economic questions. Telpak has been in litigation during much of its ten-year history, most recently concerning the reasonableness of the carriers' permitting certain classes of customers to share, while barring others from doing likewise. In 1970 the FCC found that this restricted sharing was unreasonably discriminatory, and ordered unlimited Telpak sharing; after partial reversal by the Court of Appeals, the Commission modified its order to require only that the unlawful discrimination be removed—leaving the carriers free to permit unlimited Telpak sharing, abolish sharing altogether, or change the whole nature of the Telpak service.[14] At this writing the outcome is unclear, but it seems likely that even though Telpak as we have known it may cease to exist, competitive pressures will dictate that a bulk-rate private line offering of some sort remain.

Provision of data processing services by communications carriers

The most significant results of the FCC Inquiry were with respect to the very controversial issues of market entry and industry structure in the computer services and communications fields. This has two aspects. First, as communications common carriers began to enter the computer and information services field in the mid-60s, the FCC realized that it must face questions of possible unfair competition. Should the carriers be permitted to engage in data processing service activities; and if so, under what ground rules? On the other side of the fence, unregulated firms in the computer services industry had expressed their desire to offer computer-based store-and-forward message switching services, both on a stand-alone "pure" basis and also as a "hybrid" system combined with various data processing functions. All such attempts were rebuffed by the carriers—to whom message switching is merely another form of communication service, long regarded as an exclusive common carrier activity. Perhaps, the Commission speculated, this policy should be modified in light of today's changing technology.

There are several frameworks within which one might consider common carrier participation in the data processing services area. The first of these, and perhaps the most obvious, is simply that they be permitted to enter without restriction of any sort. This assumes that data processing services will remain an unregulated activity—which, as discussed above, is the way the FCC has left the matter. The common carriers would then be (1) offering both a regulated monopoly communication service and an unregulated highly-competitive data processing service, and (2) using common plant and personnel, at least to some degree, because of the economies that would be realized in that way.* Both factors would lead to problems of cost separation and allocation.

The FCC and the state public utility commissions in the U.S. establish rate and profit ceilings for common carrier services. This is done by setting rates for communications services at a level which will permit the carrier to recover his operating costs for the service, and earn a profit equal to a fixed percentage or percentage range (established by the regulatory commission) of his plant investment, or "rate base," associated with the service. This approach, called "rate base regulation," encourages the carrier to capitalize as many of his costs as possible, so as to increase the size of the rate base.

For the competitive (unregulated) data processing service, on the other hand, the carrier's incentive is to minimize the cost charged to that service. One then sees the immediate possibility for an intentional or covert type of cross-subsidization in which certain costs properly attributable to the data processing service would, in fact, be charged to the communications service—forcing the communications user, who has no choice in the matter, to bear a portion of the carrier's cost of entry into the data services market. The avoidance of this problem would require stringent cost separation techniques, and would require allocation of joint costs in the case of common use of facilities or personnel. These might be possible to accomplish, but only with some difficulty and with certain questions remaining.

* There is disagreement as to the importance of any economies which might be achieved by using common facilities for both communications and data processing services.[29] Western Union has urged that it be permitted to utilize off-peak capacity in its switching computers for data processing services, but some observers have estimated that any savings from such a practice would have a negligible effect upon WU's overall costs.[3]

One possible solution which had been proposed[21] would be to require that a common carrier offering data processing services do so under the framework of a separate corporate subsidiary which would not share resources with the carrier, thereby avoiding many of the cost separation problems. To avoid cross-subsidization from one service to another, the data processing subsidiary would be charged full price for the communication lines that it might require for a multi-access service, the same as any other user would pay. This would eliminate some of the potential problems of cross-subsidization. However, an additional form of cross-subsidization could occur if the parent corporation were buying data processing services (either machine time rental, or programming and consulting services) from the subsidiary. Intentional over-charging of the common carrier for these services would be just another way of the subsidiary's receiving undeserved revenue from the carrier's communications customers. To prevent this from occurring, the carrier could be barred from obtaining data processing service from its affiliate.

In its final decision in the computer inquiry, the FCC accepted these proposals, and adopted rules providing that "common carriers desiring to provide data processing services [can] do so only through affiliates utilizing separate books of account, separate officers, separate operating personnel, and separate equipment and facilities devoted exclusively to the rendition of data processing services." The Commission also prohibited the data processing affiliate from using the carrier's name and barred the carrier from obtaining any data processing services from the affiliate.[12] (Small carriers with annual revenues of less than one million dollars were exempted from these prohibitions.) Several independent telephone companies appealed this decision to the courts, and other parties petitioned the FCC for reconsideration. Pending disposition of these matters, the U.S. Court of Appeals stayed the effectiveness of the FCC's order. At the time of this writing, no further action has been taken, so the outcome remains unclear. The Commission has shown no inclination to substantially change its order, however, and it appears doubtful that the carriers will be able to convince the court that the Commission acted in an "arbitrary and capricious" manner—which would be necessary in order for their appeal to succeed. The likelihood thus is that the thrust of the 1971 order requiring complete separation of common carrier communications and data processing activities will remain.

Hybrid message-switching services

The second aspect of the market entry question studied in the FCC Inquiry regarding computer and communication services concerns those services which combine both data processing and communications (message-switching) functions in a single "hybrid" package. Should the provision of such computer-based services be treated as a common carrier undertaking and regulated accordingly, or would regulatory forbearance be more appropriate?

Message-switching is a relatively old communications concept. Most readers are familiar with the old punched paper tape reperforator systems sold by AT&T and Western Union. Western Union's Public Message Service (the common telegram) also uses a paper tape reperforator network. The only thing new in the message-switching concept is the fact that we are now using general-purpose digital computers to perform the switching function, and these same computers may also perform data processing functions—yielding a hybrid remote-access system which offers both communications and data processing services to its users. Since the FCC decided in the Inquiry that data processing services would not be regulated, whether using communications lines or not, and since the Commission considered and rejected the possibility that it might "de-regulate" pure message-switching services, how should the in-between hybrid service be treated?

One possibility was that the entire hybrid service could be exempted from regulation, even where it is oriented essentially to satisfying communications (message-switching) requirements of the subscriber. Some of the parties in the Inquiry suggested this course on the grounds that to do otherwise might retard the development of valuable hybrid services—which the carriers might not choose to offer, if given an exclusive monopoly in this area—and also that such computer-based services had few, if any, of the characteristics of traditional common carrier undertakings requiring government regulation (natural monopoly, need to make service widely available to the public, etc.). Rather, the argument ran, the hybrid service would flourish best in an unregulated competitive environment, regardless of the mix of data processing and communications features it might exhibit. The FCC rejected this argument, for two reasons: (1) it felt that there was a real question whether it had the authority under the Communications Act to decline to regulate a public communications service after Congress has decided that all such services are subject to regulation, and (2) it concluded that, based on the record in the Inquiry, those hybrid services which are "essentially communications" in nature "warrant appropriate regulatory treatment as common carrier services under the Act."[12] The Commission did not explain how it reached this latter conclusion, but one might assume that it did so merely because the proponents of change failed to convince it that the traditional regulatory treatment of communications services was inappropriate in this case.

On the other extreme, some of the common carrier respondents in the Inquiry urged that the Commission is obligated by law to regulate the hybrid service insofar as it contains a communication component. This view was rejected as well, for the Commission felt that the imposition of regulatory constraints on what is essentially a data processing service, just because it contained some incidental communications elements, would inhibit flexibility in the development of such services and would be contrary to the public interest. Furthermore, the Commission held that it has discretionary latitude "to refrain from subjecting a marginal activity to [its] regulatory process where it is clear that the public interest will be served by such a course."[12]

Having rejected both extremes—to regulate or to decline to regulate all hybrid services, regardless of their primary orientation or purpose—the FCC adopted a middle position calling for regulation of only those hybrid services which are essentially communications-oriented and in which data processing plays just an incidental part. This approach is unfortunately much more difficult to administer, and requires a case-by-case examination of each service to determine on which side of the line it falls. Perhaps even more important, the dividing line itself must be clearly defined by the Commission, in order to provide advance guidance to the entrepreneur and system designer and to avoid endless litigation.

The Commission made an attempt to define a clear line between regulated and unregulated hybrid systems, but it is questionable whether this guidance will suffice. First it said that,

> "*where message-switching is offered as an* integral *part of and as an* incidental *feature of a package offering that is primarily data processing, there will be total regulatory forbearance with respect to the entire service whether offered by a common carrier or non-common carrier, except to the extent that common carriers offering such a hybrid service will do so through* [*separate*] *affiliates. . . . If, on the other hand, the package offering is* oriented essentially *to satisfy the communications or message-switching requirements of the subscriber, and the data processing feature or function is an* integral part of and incidental to *message-switching, the entire service will be treated as a communications service for hire, whether offered by a common carrier or a non-common carrier and will be subject to regulation under the Communications Act.*"[11] (*emphasis added.*)

Two tests were given to help apply this criterion:

> (1) Does the service, by virtue of its message-switching capability, have the attributes of the point-to-point services offered by conventional communications carriers, and is it basically a substitute therefor? It so, this suggests that regulation may be applicable.
>
> (2) Does the message-switching feature of the service facilitate or relate to the data processing component, or are the two components essentially independent?

In order to avoid regulation, the message-switching feature must be not only secondary in importance as compared with the data processing aspect of the service, but also the two must be closely related to one another. If the two components are functionally independent of one another (although perhaps performed on the same computer), the FCC's staff has indicated that it would regulate the entire service—regardless of the preponderance of the data processing element.

Considering this second test, it is interesting to note a certain circularity in the Commission's reasoning. If a firm desires to offer the public a hybrid service having essentially unrelated data processing and message-switching components, this test would require that the service be offered only on a regulated common-carrier basis. The firm *could* perhaps convince the FCC to grant it a certificate of public convenience, interest, and necessity so that it could do business as a common carrier, but then it would encounter another snag. As a common carrier, the firm (if it or its corporate affiliates had annual revenues of $1 million or more) would be barred from offering data processing services except through a wholly separate affiliate. Therefore, the proposed hybrid service could never be offered—the new carrier could not provide data processing, its unregulated affiliate could not provide communications service, and the two could not collaborate in a joint "package" offering. If this is the result the Commission really intended, it would have been better for it simply to have stated that a hybrid service can be offered *only* when the two components are closely related and integral to one another—and that the service will then be regulated or not depending upon which component is preponderant.

We should also note, however, that the FCC will undoubtedly apply a *de minimis* rule, which will as a practical matter exempt from regulatory control those systems having a non-integrally-related communications element which theoretically makes them subject to regulation, but which is just too small or insignificant for the Commission to worry about. The chances of such a system escaping the regulatory net are probably greater where the system operator attaches little importance to the communications-for-communications'-sake capability of his system, in his promotional literature, advertising, and so forth. Thus, a remote-access computer service in which the central processor spent 99 percent of its useful time doing data processing and 1 percent of its time doing unrelated message-switching for system subscribers could probably be offered to the public by a private firm without FCC objection, but if the supplier placed a promotional emphasis on the communications aspect of the system it would become more likely that the common carrier providing communication circuits for the system would object and the FCC would be forced to rule on the case.

The FCC has emphasized that it considers the whole field of hybrid systems as being in its infancy, and that it (the Commission) will not hesitate to modify its rules regarding such systems as they mature and perhaps require different treatment. For this reason, we should not regard the above-stated guidelines as forever frozen in concrete, but it is still appropriate to consider whether their effect will be beneficial or detrimental in the near future.

Perhaps the most serious shortcoming of these rules is their vagueness. Just what is "integral" and "incidental," and what is not? How does one determine which component is the dog and which the tail in a hybrid system containing large doses of both data processing and message switching? As a procedural matter, can a system designer obtain a definite FCC ruling in advance (or, indeed, would he want to disclose his plans and thus forewarn his competitors?), or must he risk his capital by building the system and hoping that he does not encounter regulatory difficulties at a later date?

The above guidelines regarding hybrid systems were tentatively established by the FCC in 1970 and finalized in 1971. Since then there have been no formal FCC proceedings involving hybrid systems, and few, if any, informal requests to the Commission's staff for advice or opinion concerning a specific proposed system or service. Yet the author is personally aware of at least two instances since 1970 in which data processing-oriented firms have contemplated implementing hybrid computer-communications services and have backed away because of the regulatory uncertainty involved. One can only wonder how many other innovative new hybrid services have been stopped before they began by the same uncertainity. Since the investment required to implement a sophisticated remote-access system is high, and the time delay and cost involved in an adjudicatory proceeding at the FCC is substantial (often consuming several years and several hundred thousand dollars), it is easy to see why the entrepreneur is hesitant to go ahead without knowing what lies in store for him. And this hesitation is especially true with respect to the smaller firms in the computer services industry, from whom might otherwise come some of the most innovative and socially-desirable computer communication services.

In summary, the FCC has addressed a broad set of issues in its Inquiry into the interrelationship of computers and communications. Its initiation of this Inquiry was a very farsighted approach to policy determination in an era of new and rapidly changing technology, and the outcome was in general quite beneficial to the computer communications community. The Commission could have taken some positive steps in the Inquiry to help insure the availability of adequate data transmission services and facilities, but this failure to act seems less serious than the inadequate "resolution" of the question of the regulatory status of hybrid computer-communications services. Hopefully, additional steps will be taken in the coming years to remove the uncertainty presently surrounding such services and to encourage the development in a competitive market of creative new services of this kind.

INTERCONNECTION

One of the regulatory policy issues which has generated considerable discussion in the past half-decade concerns the interconnection to the telephone network of privately-owned terminal devices and communication systems. Historically, the telephone companies have generally prohibited such interconnection by their subscribers, which might jeopardize the telephone network if uncontrolled and almost certainly would cut into carrier equipment-rental revenues. Until recent years the carriers' well-developed arguments of potential harm have been accepted without question by the FCC and the various state utility commissions, which permitted the carriers to build a seemingly-impregnable wall around the telephone network; but starting in 1956 the wall began to crumble.

A decade before the famous *Carterfone* ruling, the legality of the carriers' blanket foreign attachment/interconnection prohibition was tested before the U.S. Court of Appeals, in *Hush-a-Phone Corporation* v. *United States*, 238 F.2d 266 (D.C. Cir. 1956). This case concerned a cuplike rubber device designed to be attached to the microphone portion of the telephone handset to provide privacy in conversation; its use on the dial network had been barred by the foreign attachments rule. Reviewing the tariff in question, a prior version of the AT&T interstate toll telephone tariff, the court found it illegal and held specifically that this ban was ". . . an unwarranted interference with the telephone subscriber's right to use his telephone in ways which are privately beneficial without being publicly detrimental." AT&T was ordered to revise its tariff to permit use of the Hush-a-Phone device, and did so—but retained the *general* foreign attachment prohibition in the tariff. Three years later the Carterfone struggle began.

The Carterfone case

The Carterfone is an acoustic/inductive device for interconnection of the base station of a mobile radio system (or other private communication system) with the dial telephone network. The Carter Electronics Corporation of Dallas, Texas, which developed the Carterfone, sold approximately 3,500 of these devices in the United States and overseas between 1959 and 1966. The Bell System and the General Telephone System warned Carter's customers that their tariffs prohibited devices such as the Carterfone on the telephone network, and that customers who violated these tariff provisions risked having their telephone service terminated. Finally, in 1966, Carter brought an antitrust suit against the Bell System and the General Telephone Company of the Southwest. The U.S. District Court in Texas referred the case to the Federal Communications Commission because the Commission has primary jurisdiction in interstate communications matters.

AT&T and GTE argued before the FCC that use of the Carterfone violated their tariffs, and presented several technical arguments to support their position that the integrity of the telephone system necessitated the use of only carrier-supplied attachments, whether acoustically coupled or directly wired to the telephone network.[21] The FCC was unpersuaded by the telephone companies' arguments and, in a unanimous opinion issued in June, 1968, found that the tariff restrictions "are, and have since their inception been, unreasonable, unlawful, and unreasonably discriminatory under the Communications Act of 1934." The Commission further concluded:

> "*. . . a customer desiring to use an interconnecting device to improve the utility to him of both the telephone system and a private radio system should be able to do so, so long as the interconnection does not adversely affect the telephone company's operations or the telephone system's utility for others. A tariff which prevents this is unreasonable; it is also unduly discriminatory where, as here, the telephone company's own interconnecting equipment is approved for use. The vice of the present tariff . . . is that it prohibits the use of harmless, as well as harmful devices.*"
>
> * * *
>
> "*In view of the unlawfulness of the tariff, there would be no point in declaring it invalid as applied to the Carterfone and permitting it to continue in operation as to other interconnection devices. This would also put a clearly improper burden upon the manufacturers and users of other devices. The appropriate remedy is to strike the tariff and permit the carriers, if they so desire, to propose new tariffs which will protect the telephone system against harmful devices, and they may specify technical standards if they wish.*"[15]

The carriers initially responded to *Carterfone* by seeking FCC reconsideration; however, this was not granted. AT&T and GTE then filed for judicial review but, realizing this was fruitless, shortly withdrew their appeals and began to revise their interstate tariffs to conform with the FCC's decision.

Tariff revisions following Carterfone

The revised tariffs permitted the attachment of customer-provided devices, such as data modems, to the dial telephone network; they also permitted the interconnection of customer-provided communication systems, such as private branch exchange (PBX) switchboards, to both the dial telephone network and to "private line" channels leased from the carriers. Three new restrictions were imposed, however:

(1) The tariffs require that the power and spectral energy distribution of signals entering the switched network from interconnected customer equipment stay within prescribed limits. These criteria are intended to protect network services from excessive noise, intelligible crosstalk, and other forms of interference as well as to minimize circuit interruption, disconnections, improper billing, and voltages which might be hazardous to line maintenance personnel.

(2) A "protective connecting arrangement" supplied by the telephone company, for several dollars per month, is required where customer-provided devices or systems are interconnected to the dial telephone network. This coupler, called a data access arrangement (DAA) or voice access arrangement (VAA), effectively isolates the line from hazardous voltages potentially generated by customer attachments, and ensures that proper signal levels are not exceeded.

(3) The tariffs require that telephone-company-sup-

plied equipment perform all "network control signaling" functions on the dial telephone network, such as dialing and hook switch connect/disconnect. An exception to this rule became effective in 1971, to permit the use of customer-owned Touch-Tone dialing units (but not conventional rotary dials). This requirement is intended to prevent improper network signaling which may result in frequent wrong numbers, improper billing, and wasted maintenance and administrative effort.

The NAS report

Many users and potential suppliers of telephone equipment objected to the new tariffs filed by AT&T, on the grounds that they are still too restrictive by requiring both telephone company-supplied protective couplers and rotary dial equipment, and did not reflect the spirit of the *Carterfone* decision. In response to these objections, the FCC initiated informal engineering and technical conferences to assist in resolving questions raised by the tariff revisions and to ascertain the desirability and feasibility of further tariff changes. The informal conferences were conducted under the impartial auspices of a panel appointed by the Computer Sciences and Engineering Board of the National Academy of Sciences. The panel reported its findings in mid-1970.[26]

The National Academy of Sciences (NAS) panel reached the following conclusions:

(1) Uncontrolled interconnection can cause harm to personnel, network performance, and property.

(2) The use of protective couplers and signal level criteria is an acceptable way of assuring network protection; however, the added equipment reduces overall reliability while increasing costs.

(3) A program of standards and enforced certification of equipment would be another acceptable way of assuring network protection. This would increase the opportunities for innovation by users, without significantly impeding innovation by carriers. According to the panel, an equipment manufacturer could not be relied upon to certify its own equipment, thus requiring certification by an independent organization or government agency. The panel further concluded that equipment certification alone is inadequate; licensing of installation and maintenance personnel would also be required.

In essence, the NAS panel concluded that either of two approaches could assure a satisfactory degree of network protection:

(1) The use of protective connecting devices supplied by the telephone companies; or

(2) A program of standardization and properly enforced certification of equipment, plus licensing of installation and maintenance personnel.

A subsequent report to the FCC, prepared by Dittberner Associates, a Washington-based firm of data communications consultants, concluded that manufacturers of data modems and other sorts of interconnected customer equipment could easily build into their equipment the necessary circuitry to protect the telephone network.[4] It further found that the carriers need not have the exclusive right to provide network protective couplers. The report also concluded that a program of standards and certification of installation and maintenance *organizations* (rather than licensing of individual persons) would be an inexpensive way to extend interconnection privileges without harm to the common carrier network.

Recent interconnection developments

Both the NAS and Dittberner reports to the FCC concluded that safe and reliable interconnection of subscriber equipment could be accomplished without need for the onerous carrier-supplied access arrangement or coupler. The FCC had received many complaints from users and equipment manufacturers that the access arrangement represented a serious burden upon them—long delays were often encountered in obtaining a unit, once installed it sometimes degraded the signals passing through it, Bell kept changing its interface specifications, the device was grossly overpriced, etc.—and the Commission was eager to begin to implement an alternative program of equipment standards and certification. To get its feet wet, the Commission chose the private branch exchange (PBX) area for initial consideration, with the hope that experience gained there could soon be transferred to other interconnection areas (such as data communications) as well.

In March, 1971 the Commission established a PBX industry advisory committee with some thirty members representing carriers, equipment manufacturers, and users, and charged it with making recommendations concerning standards applicable to customer-provided PBX equipment. The PBX advisory committee intially proposed that the FCC establish a permanent Interconnect Board to oversee standards-making, equipment testing and certification, and other administrative aspects of an expanded interconnection program. This plan was criticized as being overly complex and bureaucratically cumbersome, and also several of the state utility commissions objected to Federal control of standards for installation, maintenance, or inspection of customer-owned equipment, or the licensing of inspectors for such equipment—feeling these to be matters of primarily local concern. NARUC, the national association of state regulatory commissioners, proposed the establishment of a federal-state joint board to provide policy guidance in interconnection matters; the FCC agreed to this proposal and promised to refer to the joint board for its consideration the final recommendations of the PBX advisory committee. Meanwhile, the PBX committee continued to explore feasible technical standards and less complex procedural machinery which might be established, and set April 1, 1972 as a target date for completion of the first phase of its final report.

At the time of this writing there is optimism that the FCC's PBX industry advisory committee will make significant headway in developing a workable interconnection program which will give maximum freedom to the PBX manufacturer and user without compromising the security of the telephone network. Such a program, if successful, can then serve as a model for other types of interconnection, such as data communications equipment. Unfortunately, the state commissions appear to represent a potentially significant obstacle to the liberalization of interconnection regulations. Commenting on the PBX advisory committee's equipment certification concept, NARUC expressed its traditional populist bias and said, "The liberalized interconnection of customer-provided PBX equipment will benefit a very affluent class of users by reducing their communication costs and, further, will benefit unregulated manufacturers of PBX equipment by increasing their profits." Perhaps NARUC should consider that communications represents a major part of the cost of doing business today, and when that cost is reduced, everyone benefits—the consumer as well as the corporation which sells to him. Also worth noting are AT&T's own admissions that the increased competition in communications—primarily through interconnection and the advent of specialized common carriers—has been good for Bell and good for the public.[28]

The changes brought on by *Carterfone* have been truly revolutionary, both in terms of creation of a new "interconnect" equipment industry and stimulation of innovation and lower costs in data communications hardware. It is estimated that nearly 200 firms have entered the data modem market since the FCC decision, bringing with them in many cases technology developed for defense and aerospace applications which previously had no commercial outlet. A wide variety of new, improved, and less expensive products have been introduced. Data terminal and multiplex manufacturers can now reduce overall system costs by incorporating modems (which they can manufacture themselves or obtain in the OEM market) directly into their equipment.[21]

The carriers have responded to this new competition by developing new products and cutting prices—much to the benefit of data communications users. For example, the "standard" low-speed modem has been Bell's series 103 data set, which has originate/answer capability (although many users operate in an originate-only mode) and rents for approximately $25 per month. When independent suppliers began to offer originate-only units for a small fraction of that price, Bell responded by introducing the 113 data set—originate-only, for roughly half the 103 price. Interestingly, 103 units in the field were then "converted" to 113s by merely snipping a lead to disable the unneeded "answer" capability. In the same vein, Bell in late 1971 announced plans to cut rental rates by 35 percent on its medium-speed series 200 data sets.

In spite of this vigorous competitive response by the telephone company, there is every indication that the interconnect industry is just beginning to take off. In the first 33 months since January, 1969 when Bell's new interconnection tariffs went into effect, Bell installed 8,100 data access arrangements for customer-owned modems—with almost 3,600, or 44 percent, of them put into service in the last six months of the period alone. The further regulatory changes which will hopefully be forthcoming soon through efforts such as those of the FCC's PBX advisory committee can only spur this growth and make available to more users the benefits of competition in the communications equipment industry.

SPECIALIZED COMMON CARRIERS

When one mentions "introduction of new competition" in the common carrier communications arena, probably the first thing which comes to mind is *Carterfone*, which introduced competition into the terminal equipment portion of the telephone network. But equally important for the future of data communications is a decision by the FCC the year after *Carterfone*, authorizing the first new "specialized carrier" to compete with the backbone intercity telephone network itself. This new carrier was Microwave Communications, Inc. (MCI), which had applied to the FCC in 1963 for permission to construct a microwave radio relay system between Chicago and St. Louis in order to offer the public a variety of dedicated private-line communications services in direct competition with Bell and Western Union.

MCI argued persuasively before the FCC that the established carriers were not adequately meeting all the specialized needs of business and data communications users and that its proposed services, tailored to these needs, would be in the public interest. On the other hand, the established carriers contended that FCC approval of a competing long-haul carrier would result in a wasteful duplication of facilities and would impair their ability to achieve maximum economies of scale. Furthermore, they said, MCI was just a "cream skimmer" trying to reap the profits of a dense high-traffic route while leaving the established carriers to serve the boondocks; if this were permitted it would destroy the traditional average-cost pricing system for telephone service and force the rates on less dense routes to rise. After almost six years of delay, hearings, legal briefs, and more delay, the FCC rejected the established carriers' arguments and approved the new entrant.[8] MCI was given construction permits for its 263-mile microwave system, and after some further procedural delays the company finally constructed its system and went on the air as a full-fledged carrier on January 1, 1972.

If this were all that happened, there would be little cause for more than academic interest among most data communications users. But the FCC's approval of MCI triggered a flood of over 1900 new microwave-station applications by several dozen firms proposing to build more than 40,000 miles of new specialized carrier communications facilities throughout the country. All but one of the applicants (16 of whom are affiliated with MCI) proposed MCI-like analog microwave facilities which would offer a variety of business-oriented private-line communication channels as well as channels specifically designed for data transmission. The exception was the Data Transmission Company, or Datran (a subsidiary of University Computing Co.), which proposed to build a nationwide all-digital switched network to offer exclusively data transmission services on both a switched and private-line basis.

These applications presented the FCC with a major policy problem, for the Commission was not sure that its *MCI* precedent should be extended on a nationwide basis without further analysis of the effects of such widespread competition among carriers. So the Commission in July, 1970 instituted a public inquiry into the merits of the specialized carrier concept. As might have been anticipated from previous experience in the computer inquiry, in which numerous respondents complained about the inadequacy of existing carrier services for data transmission, the specialized carrier inquiry elicited strong support for the MCI and Datran concepts from all sectors of the computer industry and from data communications users. The established carriers, of course, continued to bitterly oppose the new competition. The same issues concerning public need, effects upon telephone rate-averaging, and duplication of facilities were debated and resolved by the FCC in the same manner as it had done in the original *MCI* case. In May, 1971 the Commission issued its decision,[13] permitting virtually free entry of all financially and technically qualified applicants for specialized carrier service (many of whom had filed for the same routes and thus would compete with each other as well as with Bell and Western Union), subject only to resolution of any radio-frequency interference with existing microwave systems.

At the time of this writing a second specialized carrier has been granted construction permits by the FCC, for private-line service between New York City and Washington, D.C., and other systems will be cleared shortly. By the end of 1973 MCI-type specialized carrier systems could be operational throughout the country, and by the end of 1974 Datran could have its digital network on the air. It is inevitable that as the various specialized carriers construct their regional microwave systems they will interconnect with each other in order to offer through service between different parts of the country, and thus a nationwide "second network" will come into being to compete with the telephone network. In addition to the microwave relay facilities which will form the backbone of this new network (just as microwave is the backbone of today's long distance telephone network), other types of communications facilities may also be used; domestic satellites may provide economical long-haul trunk circuits (note that two specialized carriers, MCI and Western Tele-Communications, Inc., have applied for permission to operate domestic satellite systems) and two-way CATV systems may be used for local distribution to subscribers in urban areas.

In terms of impact upon data communications, the advent of the specialized carrier will bring with it a number of important and generally beneficial results:

- New data communication services (many different channel sizes, rapid-connect switched service, etc.).
- Lower error rates in data transmission.
- Specialized carrier rates considerably lower than today's telephone rates for equivalent service.
- Modification of certain telephone rate structures, as a response to the new competition.
- More rapid introduction of new technology (e.g., advanced digital microwave and solid-state switching) by the new and old carriers alike.
- More responsiveness by the telephone companies to the needs of data communications users.

- Ability of the FCC to use the new carriers as a yardstick to measure the performance of Bell and the other telephone companies.

The degree to which the individual user will be affected by or able to take advantage of these results will obviously vary widely depending upon individual circumstances, but it is safe to predict that the overall impact of the specialized carriers will be substantial.

Due to limitations of space this has been a necessarily brief and superficial overview of a rather involved subject. The reader interested in a comprehensive discussion of the specialized carrier concept, specific features of the MCI and Datran systems, policy considerations, and outlook for the future is referred to recent papers by the author in References 30 and 31.

DOMESTIC SATELLITES AND CABLE TELEVISION

During the past several years the FCC has taken action to open the communications market to the entrance of new competitors and to make available new services to the public in two important areas: domestic communications satellites and cable television (CATV). Both of these developments hold great potential significance for data communications.

Domestic communications satellites

The world's first satellite, Sputnik I, was launched fifteen years ago. Congress formed the Communications Satellite Corporation (COMSAT) ten years ago, to serve as the U.S. agent in establishing and operating an international communications satellite network. The first operational satellite in this international network was launched seven years ago, and in the same year the first of many proposals was made to establish a *domestic* communications satellite network for the U.S. Today. Russia has such a network (albeit with random-orbiting rather than synchronous satellites). Canada will launch its first domestic satellite later this year and other nations are also moving toward a similar goal. Still the U.S. does not have a domestic satellite system or even approved plans to build one. At last, however, it appears that one or more firms will be given FCC approval to establish such a system or systems in this country.

The economic benefits of a domestic communications satellite system were appreciated years ago,* and the technology to implement such a system has also been available for five years or more—for a domestic system would use essentially the same sort of geostationary synchronous satellite which INTELSAT has been using in its international network since 1965 (quantum improvements in system performance, cost, and channel capacity have been made since 1965, and several new generations of satellites have appeared, but the concept has remained unchanged). The stumbling block which has kept the U.S. from taking advantage of this technology has been a series of unresolved fundamental policy questions such as whether there should be only one or more than one domestic system, who should be permitted to operate such system(s), whether the satellite operator should sell service directly to the public or through the established terrestrial carriers (such as COMSAT does in the international arena), and other issues.

After the American Broadcasting Company submitted to the FCC in 1965 a semi-serious proposal to build a domestic satellite system for low-cost television distribution, the Commission opened an inquiry on this subject and invited comments and counter-proposals. Several alternative satellite proposals were promptly submitted by other organizations, and considerable public interest was generated. The FCC began its deliberations and for a while it looked as if we would have a satellite system in operation by 1970 or 1971. But then the Commission's decision-making process ground to a halt for more than a year while President Johnson's Task Force on Communications Policy studied the issues in late 1967 and 1968. The Task Force eventually recommended approval of a single "pilot" program to be run by COMSAT,[27] much as COMSAT had proposed in 1967 (see Figure 1), and the FCC—predisposed in this direction anyway—began to move once more toward a decision along these lines.

But President Nixon took office a few weeks after the Task Force report was received, and before the FCC had time to act the White House requested another delay so that the President's staff could study the issues anew. This study was directed by Dr. Clay T. ("Tom") Whitehead—a White House staff assistant who was

* For the data communications user, these benefits include a distance-independent cost structure, with circuit costs only a fraction of corresponding terrestrial microwave or cable costs for long-haul circuits—those over a thousand or so miles in length; a potential wide variety of channel bandwidths and data transmission capacities; lower error rates than frequently experienced on long terrestrial circuits; and relatively easy and economical access to remote areas of the country which may lack adequate terrestrial transmission facilities.

	1957	Sputnik I launched by USSR—the world's first satellite.
	1962	Congress passes Communications Satellite Act, providing for establishment of a new privately-owned corporation, Communications Satellite Corporation (COMSAT), to serve as the U.S. entity in *international* satellite communications.
	1963	Syncom launched by NASA—the first geostationary synchronous satellite.
	1964	International Telecommunications Satellite Consortium (INTELSAT) formed to create international satellite communications network.
	1965	Early Bird launched—the first commercial communications satellite and the beginning of the INTELSAT network.
	1965	American Broadcasting Co. submits proposal to FCC for a domestic TV-distribution satellite.
	1966	FCC opens inquiry on domestic satellites, and asks broad policy questions regarding establishment of systems by non-government entities.
	1966	Ford Foundation submits counter-proposal for a multi-purpose domestic satellite, with profits to be used to support educational television.
	1966	COMSAT proposes a multi-purpose domestic satellite, to be operated by COMSAT as a "direct-to-end-user" carrier (*not* a "carrier's carrier," as COMSAT operates internationally). Four other organizations also propose satellite systems.
March	1967	COMSAT proposes "pilot demonstration program," with two satellites to be operated by COMSAT as trustee until FCC decides ownership issue.
August	1967	President Johnson appoints Task Force on Communications Policy to study domestic satellites and other issues; FCC suspends action in its domestic satellite inquiry pending receipt of Task Force recommendations.
December	1968	President's Task Force submits report recommending approval of a single "pilot" domestic satellite program, with COMSAT having primary responsibility.
February	1969	General Electric Co. proposes domestic satellite concept using time-division multiple access (TDMA) techniques to provide new and expanded services.
July	1969	As FCC prepares to approve a pilot domestic system substantially as recommended by President Johnson's Task Force, the White House requests a delay until President Nixon's staff can study the matter and submit recommendations.
January	1970	White House sends memo to FCC urging approval of all financially and technically qualified applicants for common carrier or private domestic satellite systems—instead of single pilot system as contemplated by FCC.
March	1970	FCC issues Report and Order in domestic satellite inquiry, adopting White House recommendation and inviting all interested parties to submit domestic satellite proposals.
March	1971	Before its deadline, FCC receives eight applications for satellite systems.
May & July	1971	FCC receives comments and reply comments from the applicants and other interested parties regarding the eight applications.
Fall	1971	NASA performs technical evaluation of the applications for FCC.
Early	1972	FCC decision expected.

Figure 1—Domestic satellite chronology

later to become the first director of the new Office of Telecommunications Policy (OTP) in the Executive Office of the President—and came to conclusions quite different from those of President Johnson's Task Force and the FCC. Rather than supporting the concept of a single pilot system to test and demonstrate the workability of domestic satellite communications, which would create at least a temporary and perhaps a permanent monopoly structure in this new field, the White House in January, 1970 recommended to the FCC a policy of immediate open entry and free competition. In summary, it was urged,

> *"Subject to appropriate conditions to preclude harmful interference and anti-competitive practices, any financially qualified public or private entity, including Government corporations, should be permitted to establish and operate domestic satellite facilities for its own needs; join with related entities in common-user, cooperative facilities; establish facilities for lease to prospective users; or establish facilities to be used in providing specialized carrier services on a competitive basis."*[16]

Although theoretically the FCC is an independent regulatory agency, and need answer only to Congress for its policies, in practice it is difficult to ignore such direct pressure from the White House. Therefore, within three months after receiving these recommendations the Commission changed course and issued a report and order adopting the concept of open entry and free competition in domestic satellites.[6] Since the satellite applications submitted back in 1965-67 were technically out-of-date by 1970, the Commission invited the submission of new applications by all organizations interested in building either common carrier or private domestic satellite systems. Before its deadline in early 1971 the Commission received eight applications for satellite systems, all to be used to provide various types

Applicant	Operational satellites	In-orbit spares	Transponders per satellite[1]	Total no. of transponders[1]	Transmit-receive earth stations	Receive-only earth stations
AT&T/COMSAT Corp.	2	1	24	48	5	0
COMSAT Corp.	2	1	24	48	4	1
Fairchild Hiller Corp.	1	1	120	120	6	0[2]
Hughes Aircraft Co./General Telephone & Electronics Corp.	2	0	12	24	6	7[2]
MCI Lockheed Satellite Corp.	1	1	48	48	20	0
RCA Globcom/Alascom	1[3]	1	12	12[3]	13[2]	0[2]
Western Tele-Communications, Inc.	2	0	12	24	4[2]	2[2]
Western Union Telegraph Co.	2	1	12	24	7	6

Figure 2—Proposed domestic satellite systems

[1] Each transponder has a capacity of some 900 one-way telephone channels, or one color television channel, or some 35 megabits/sec. of time-division multiplexed (TDM) data.

[2] Plans include numerous additional earth stations of this type in the future.

[3] Initial system only. An additional satellite is proposed for later launch, when the first one approaches saturation.

of common carrier services. Figure 2 lists the applicants and gives a rough idea of the size of their proposed systems—number of satellites, transponders (the broadband microwave communications repeaters aboard the satellite), and earth stations. The different applicants proposed to offer a wide variety of communications services with their systems, including the following:[5]

- Augmentation of existing common carrier (AT&T, WU, GTE) trunk-capacity, for traditional switched and private-line services (voice telephone, data, teletype, facsimile, etc.).
- High-speed data channels at speeds well above the maximum available from the carriers today.
- Multi-destination one-way data channels for the distribution of news, educational material, financial, medical and scientific data.
- Video channels for studio-to-studio program distribution for the three commercial television networks and for CATV operators throughout the nation.
- Cost reductions of 50 percent and more for conventional voice, data, and video private line (leased) service.
- Public interest services such as free or reduced-rate video channels for educational television; free service to the medical community for improvement of health care; and full service to remote areas such as the Alaskan bush and the Panama Canal Zone.
- Intercity Picturephone service.
- Motion picture distribution service.
- Electronic mail service, in conjunction with the U.S. Postal Service.
- Lease of circuits or entire transponders to other carriers.

It is anticipated that the FCC will act on these applications in 1972, and will almost certainly approve several—if not all—of them. Two to three years thereafter, the U.S. will finally have an operational domestic satellite network, probably composed of a number of firms in hot competition with one another. Based on our national experience with industrial performance in a wide variety of competitive and non-competitive industries, it is reasonable to expect this competition to yield significant benefits. In the first place, it is unlikely that any single firm which might have been chosen to provide domestic satellite service on a monopoly basis would offer the wide range of services listed above, for several reasons—lack of competitive pressures, desire to protect existing terrestrial communications revenues (if the satellite firm were an established carrier), unwillingness to face the risks of introducing so many untried new services, lack of capacity of a single satellite system, etc. With multiple competing suppliers in the marketplace, the rate of innovation will be higher and the cost savings due to the new technology will be passed on to the user to a greater degree. Of course, if the market for domestic satellite service is inadequate to support all the competitors, ill effects of "ruinous competition" could occur; but it is expected that the would-be satellite operators will evaluate this risk carefully before undertaking actual system construction (and for this reason not all of the systems which the FCC might authorize may actually be built—just as with the specialized carriers, some of the weaker applicants may fall by the wayside).

In conclusion, it is unfortunate that this country, which was responsible for the development of most of today's impressive communications satellite technology, has had to wait so long to take advantage of it at home.

However, the changes in national communications policy thinking—from a monopoly environment to free competition in domestic satellites—which began to occur in 1969 should make our experience with the domestic satellite much more meaningful and beneficial. So perhaps it is worth waiting for after all.

Cable television service

When the data communications user or the designer of remote-access computer systems thinks about the communications resources at his disposal, he generally limits his universe to the various services offered by the traditional common carriers—the telephone company and Western Union. As we have seen, the new capabilities of specialized carrier systems and of domestic communications satellites have come onto the horizon in the last several years, and will have a large impact in the near future. Many users are at least vaguely aware of these developments and eager to begin to reap the benefits they hold in store. But few people in the computer community have given serious attention to yet another new technology and new communications sub-industry which has been evolving around us and which may eventually have an even larger impact in making the power of the computer widely and economically available throughout our society. This is the field of cable television, or CATV (sometimes referred to as community antenna television), which began in 1949 in the hills of eastern Pennsylvania and Oregon and has since spread to over 4000 cities and towns throughout the country.

The early CATV systems were located in isolated rural communities and consisted simply of a tall antenna (perhaps mounted atop a nearby hill) to pick up broadcast television signals from cities some distance away, amplifiers to boost the weak signals, and a coaxial cable network to distribute the signals to subscribing households in the community. Later, microwave radio began to be used to enhance the value of many CATV systems, by piping in signals from TV stations so distant that they could not be received by even the cable system's master antenna. In some isolated communities CATV provided the only means of television reception; and in other areas it enabled viewers to receive many more TV channels than they could by using a home antenna. The new medium thus offered a real benefit to the consumer and was also beneficial to the broadcaster, for it enlarged the size of his audience and thus his advertising revenues. But it was not long before cable outgrew its small-town beginnings and began to look toward the cities where the most dense and lucrative concentrations of viewers are located.

Sometimes a big-city cable system could (and can) attract subscribers by merely providing clear ghost-free reception of the local TV stations (e.g., in New York City, where ghosts are a serious problem for many viewers); but generally the cable operators found that success in the metropolitan areas depended upon their ability to "import" distant signals from other cities—for otherwise why would a viewer who already has several local channels pay $5.00 or so per month to connect to the cable? And, of course, the local over-the-air broadcasters were afraid that their previously captive audience would be fragmented if the cable system were permitted to import attractive distant signals. The stage was thus set for a regulatory battle of major proportions.

After previously declining to regulate cable television on the grounds that it was unsure of its jurisdiction under the Communications Act of 1934, the FCC changed course in 1965 and asserted jurisdiction over microwave-fed cable systems. Then in 1966, acting under pressure from the broadcasters, the Commission asserted jurisdiction over all CATV systems and imposed severe restrictions upon the carriage of distant signals in the top-100 television markets [which range from New York City (No. 1) to places like Fargo-Grand Forks-Valley City, N.D. (No. 98) and Monroe, La.-El Dorado, Ark. (No. 99)]. In 1968 the Supreme Court affirmed the FCC's assertion of regulatory jurisdiction over CATV, and soon thereafter the Commission imposed a virtual freeze on new CATV systems in the top markets—until the issues could be sorted out and a workable means found to open up cable's potential to serve the public without at the same time undermining the foundation of the existing over-the-air broadcast structure.

The Commission saw in CATV a threefold potential contribution toward improving the nation's communications system: "providing additional diversity of programming, serving as a communications outlet for many who previously have had little or no chance of ownership of or access to the television broadcast system, and creating the potential for a host of new communications services."[7] This third characteristic of cable is, of course, the important one for the data processing and data communications community and it is also the aspect of cable which has caused the most speculation about the future nature of the industry. The literature is replete with discussions of the "wired city" concept, which is based upon the broadband communications capability of the CATV cable.[25] Not only can cable provide 20, 40, or more channels of commercial, educational, and special-purpose (e.g., weather, stock ticker, news wire) television, but also it can provide a great variety of computer-oriented services such as information distribution and retrieval, interactive computing, remote shopping, computer-assisted instruction, opinion polling, and many others. In order to provide most of these services, CATV systems must have two-way transmission capability (for at least low-speed data input from the user in his home or office) rather than the one-way cable system designs which are almost universally used today. Considerable development work on two-way cable has been undertaken, and several experimental systems are presently in operation[1,19,24]—so the most critical problems today are regulatory and economic rather than technical in nature.

After freezing the growth of cable in the major metropolitan centers, and thus gaining a little breathing room, the FCC began to explore the policy problems in depth. In the fall of 1969 the Commission took an important step to further the growth and economic viability of cable, and permitted CATV systems to originate programming and to sell advertising. The key issue of distant signal importation, and a related question of compensation for program copyright owners, were the subjects of several attempted compromises between cable and broadcast interests during the period 1969-71. Finally in 1971 the FCC announced its intention to issue a comprehensive set of rules which would permit limited importation of distant signals into the major markets, would at the same time provide a measure of protection for affected broadcasters, and would require that cable systems include substantial non-broadcast features—mandatory two-way capability, free channels for public access, government and educational use, and excess channel capacity to be sold on a common carrier basis to all comers.[2,7] In late 1971 the FCC and OTP engineered a cable/broadcasting compromise agreement essentially along these lines, and at the time of this writing new FCC rules were expected momentarily.

Some observers feel that the expected FCC rules are still overly protective of broadcasters in the top-50 markets (the rules are less protective in markets 51-100), and will slow the development of cable in those cities. Thus, the Commission may need to liberalize its rules in the future after more experience is gained. In the meanwhile, at least the 1968 freeze is broken and cable can begin to enter the large cities and hopefully to fulfill the many promises which have been made about its potential.

The long-range implications of CATV for data communications and remote-access computer services are enormous. Once high-capacity two-way cable systems are operational in the major urban areas they will provide cost/effective access to the homes and offices of the customers of many of the remote-access computer services now on the drawing boards; they will provide broadband point-to-point transmission capacity for data communications users in the urban area; they will provide an additional alternative means of connecting specialized intercity microwave carriers with their customers in the city; and very likely many of the urban CATV systems will eventually be interconnected with one another through domestic satellite facilities, making possible a nationwide broadband network of enormous potential. The cable system already finds itself in ever-increasing competition with the broadcaster; soon it will also compete with the telephone company and perhaps with other sorts of communications entities as well, in providing a wide range of new communications services. Having set the stage by permitting CATV to compete, the FCC must now face the problems of regulating the new competition—setting detailed standards for common carrier services on the cable, etc.—so that CATV technology will most effectively serve the public. Questions of state regulation[33] and of possible new federal legislation will also be important in the coming years as cable matures.

SUMMARY

The past five years have witnessed a number of regulatory developments which will have a major impact upon the future of data communications. The FCC has introduced competition into the "interconnect" equipment market and into the long-haul communications carrier market, it has invited proposals for competitive domestic communications satellite systems from all interested parties, and it has cleared the way for cable television to enter the large urban areas and to provide a variety of new competitive communication services. Meanwhile, in its computer inquiry the Commission has studied several important computer-communications policy issues and has taken action regarding some of them, such as attempting to establish a regulatory framework for the public offering of hybrid message-switching services. Each of these developments is important in its own right for the benefits it brings to data communications; but viewed together they take on a larger importance, signaling the beginning of a new competitive era in the communications industry—a change in regulatory thinking which should benefit all members of the computer-communications community.

REFERENCES

1 W S BAER
Interactive television: Prospects for two-way services on cable
Rand Corp Report No R-888-MF November 1971

2 D BURCH (Chairman, FCC)
Cable television: Growth and future evolution
Testimony before the Communications Subcommittee of the Senate Commerce Committee June 15 1971

3 D D COWAN L WAVERMAN
The interdependence of communications and data processing: Issues in economics of integration and public policy
2 Bell J Econ and Mgt Sci pp 657-677 1971

4 *Interconnection action recommendations*
Dittberner Associates Report to the FCC September 1970

5 R A ELDRIDGE B M HADFIELD
M P TALBOT JR
Summary of the domestic communication-satellite applications
MITRE Corp Report No M71-96 August 1971

6 *Establishment of domestic communication-satellite facilities by non-governmental entities*
FCC Docket 16495 Report and Order ____ FCC____ (1970)

7 *Letter to the House and Senate Commerce Committees cable television*
FCC August 5 1971 FCC 71-787

8 *Microwave Communications, Inc*
FCC Docket 16509 *et al* Decision 18 FCC 2d 953 (1969)
Petitions for Reconsideration Denied 21 FCC 2d 190 (1970)

9 *Regulatory and policy problems presented by the interdependence of computer and communication services and facilities*
FCC Docket 16979 Notice of Inquiry 7 FCC 2d 11 (1966)

10 ______
FCC Docket 16979 Report and Further Notice of Inquiry 17 FCC 2d 587 (1969)

11 ______
FCC Docket 16979 Tentative Decision of the Commission 28 FCC 2d 291 (1970)

12 ______
FCC Docket 16979 Final Decision and Order 28 FCC 2d 267 (1971)

13 *Specialized carrier inquiry*
FCC Docket 18920 First Report and Order 29 FCC 2d 870 (1971)

14 *Telpak tariff sharing provisions of AT&T and Western Union Telegraph Co*
FCC Docket 17457 Decision 23 FCC 2d 696 (1970) aff'd in part & rev'd in part, *sub nom. AT&T et al* v. *FCC,* ____ F. 2d ____ (2d Cir. 1971); FCC decision on remand, ____ FCC 2d ____ (1971)

15 *Use of the Carterfone device in message toll telephone service*
FCC Dockets 16942 and 17073, Memorandum Opinion and Order 13 FCC 2d 420 (1968); Petition for Reconsideration Denied 14 FCC 2d 571 (1968)

16 P FLANIGAN (Assistant to the President)
Memorandum for the Honorable Dean Burch, Chairman of the Federal Communications Commission Jan 23 1970

17 M R IRWIN
The telecommunications industry
New York Praeger 1971

18 M R IRWIN
Time-shared information systems: Market entry in search of a policy
31 AFIPS Proc pp 513-520 FJCC 1967

19 R K JURGEN
Two-way applications for cable television systems in the 70's
IEEE Spectrum November 1971 pp 39-54

20 J MARTIN A R D NORMAN
The computerized society
Englewood Cliffs NJ Prentice-Hall 1970

21 S L MATHISON P M WALKER
Computers and telecommunications: Issues in public policy
Englewood Cliffs NJ Prentice-Hall 1970

22 A R MILLER
The assault on privacy
Ann Arbor Univ of Michigan Press 1971

23 M R MITCHELL
State regulation of cable television
Rand Corp Report No R-783-MF October 1971

24 *Urban communication systems*
MITRE Corporation Report No M71-64 November 1971

25 *Communications technology for urban improvement*
National Academy of Engineering, Committee on Telecommunications, Report to the Dept of Housing and Urban Development June 1971

26 *A technical analysis of the common carrier/user interconnections area*
National Academy of Sciences, Computer Science and Engineering Board, Report to the FCC June 1970

27 *Final report*
President's Task Force on Communications Policy Dec 7 1968 Washington DC Government Printing Office 1969

28 H I ROMNES (then Chairman of the Board, AT&T)
Toward a second century
Speech before the Telephone Pioneers of America New York Sept 21 1971

29 *Report to the FCC on Docket 16979*
Stanford Research Institute Vols 1-7 February 1969 published by the National Technical Information Service Chapter Nine of *Computer-Communication Networks* ed

30 P M WALKER S L MATHISON
Regulatory policy and future data transmission services
Chapter Nine of *Computer-Communication Networks* ed by N Abramson and F F Kuo Englewood Cliffs NJ Prentice-Hall in press

31 P M WALKER S L MATHISON
Specialized common carriers
Telephone Engineer and Management Oct 15 1971 pp 41-61

Data communications in 1980—A capital market view

by ROBERT E. LA BLANC and W. E. HIMSWORTH

Salomon Brothers
New York, New York

INTRODUCTION

Communications industry revenues presently represent approximately 2 percent of the GNP and are growing substantially faster than the GNP. The important question thus arises as to what capital resources will be available to support continued growth in the Seventies—a question of particular interest to the data communication user, whose needs are projected to expand at an explosive rate. By 1980, this growth could occur in any of the following areas:

- Continued use of the existing analog telephone network as well as new data services on the Picturephone network;
- New digital private line facilities offered by the existing carriers;
- Expansion of Western Union facilities and service offerings;
- Domestic satellite systems;
- Private line and switched facilities of the special service common carriers;
- Possible joint use of data on CATV systems.

This paper examines the capital resource needs of each of the major communications industry segments, and then, in the light of the likely expansion of the GNP and the consequent expansion of funds available from the capital markets, projects what data communication facilities are likely to be available by 1980.

While it is obviously difficult, if not impossible, to accurately project ten years into the future, we hope that the estimates made in this paper will provide a basic understanding of the problems and opportunities confronting the industry in the coming decade.

CAPITAL REQUIREMENTS TO 1980

The capital requirements of the communications industry for 1971-1980 are developed in this section for each of six market segments. Projections of operating revenue and gross plant, based primarily on industry expectations, are shown in Table I. Plant investment provides an indication of total financial requirements which, in turn, leads to our estimates of capital market demands shown in Table II.

Telephone

In terms of revenue and capital requirements, the telephone companies, and the Bell System in particular, have dominated and will continue to dominate the communications industry. The present telephone network, including the independents, serves a total of about 125 million telephones, which is up from 74 million in 1960 and 94 million in 1965. Interstate messages handled per year have increased from 1.0 billion in 1960 to 1.6 in 1965 to 2.7 billion by the end of last year. Gross plant investment has increased 134 percent in the decade of the Sixties, and today stands at over $66 billion. Annual revenues during the same period grew from $7.8 billion to $16.8 billion, or an increase of 215 percent.

In looking ahead to 1980, several factors should be kept in mind. AT&T has shown that local telephone service growth correlates closely with the growth in new family formations, and that demographic statistics indicate that with those entering the 18-to-25-year marriageable age category in the coming ten years, local telephone growth is expected to average 8 percent yearly. Toll service revenue, which grew an average of 11.5 percent yearly in the 1960s, finished the decade with growth of 15.2 percent in 1969. In 1970, during the economic downturn, network long distance revenues grew close to 8 percent for Bell. Interstate telephone message volume, excluding WATS, is expected to increase to about 10 billion messages per year by 1980 and Bell expects the number of private line circuits to more than double. We thus believe that total revenue growth

TABLE I—Revenue and Plant Projections (Billions of Dollars)

	1960	*1965*	*1970*	*1975E*	*1980E*
Operating Revenue					
Bell System	$8.1	$11.3	$17.0	$28.0	$40.0
Independent Telephone	1.0	1.7	2.9	4.7	7.3
CATV	0.0	0.1	0.4	1.0	3.1
Special Service Common Carriers	—	—	—	0.1	0.8
Western Union	0.3	0.3	0.4	0.8	1.5
Domestic Satellite	—	—	—	0.2	0.4
Gross Plant					
Bell System	24.8	36.3	54.8	90.0	130.0
Independent Telephone	4.0	7.0	12.6	21.4	34.8
CATV	0.1	0.2	0.9	2.6	7.8
Special Service Common Carriers	—	—	—	0.5	1.5
Western Union	0.3	0.5	0.7	1.5	3.0
Domestic Satellite	—	—	—	0.6	1.2

Source: Salomon Brothers' estimates.

from the telephone sector of communications can be reasonably expected to grow at an 8.5 percent to 10 percent compound rate through 1980.

Certainly, additional revenues generated by such growth areas as data communications and the Picturephone network will augment this growth. Bell has forecast growth in data transmission revenues from $500 million in 1970 to $5 billion in 1980. A Picturephone network of from 500,000 to one million terminals can be safely assumed for 1980, with data usage estimated at 25 percent of total network usage.

The telephone industry has conservatively estimated annual revenues approaching $50 billion by 1980. This would imply gross plant investment of over $160 billion in the industry by 1980. Based on our models of the industry, we estimate a need for about $63 billion in outside financing to fund this telephone plant investment. This includes roughly $5 billion in 1971, which accounted for almost all of the communication industry's capital demand.

CATV

CATV today serves over 6 million homes in over 4500 communities. Growth from 650,000 homes in 1960 and 1.3 million in 1965 has been at an annual rate of about 25 percent. With the projected implementation of the recent FCC proposals, growth to 25–30 million subscribers by 1980 is indicated. Revenues, up from an estimated $39 million in 1960 and about $80 million in 1965, reached $360 million in 1970. We believe revenues can reasonably be on the order of $1 billion by 1975 and $3.1 billion by 1980, reflecting greater channel usage and two-way services. Gross plant investment from a $900 million level in 1970 should be at about the $2.6 billion level in 1975 and $7.8 billion in 1980. This will require an estimated $5.7 billion in outside financing.

Special service common carriers

The special service common carriers, both DATRAN and the MCI types, are obviously starting from scratch. Construction permits are being issued by the FCC in accordance with the broad general policy laid out in the final order in Docket 18920. Estimates vary as to the impact these carriers will have on the industry. Forecasts of 1980 revenues range from $750 million (Arthur D. Little) to $2 billion (Frost and Sullivan). Taking a conservative approach, an estimated gross plant investment of at least $1.5 billion by 1980 and an external financing requirement of $1.2 billion is indicated.

Western Union

Western Union is rapidly moving to become the major provider of computer-enhanced record com-

TABLE II—External Financing Projections (Billions of Dollars)

	1971-75	*1976-80*	*Total*
Telephone	$28.0	$34.5	$62.5
CATV	1.4	4.3	5.7
Special Service Common Carriers	.5	.7	1.2
Western Union	.4	.7	1.1
Domestic Satellite	.4	.4	.8
	30.7	40.6	71.3
Refinancing	.5	4.8	5.3
	$31.2	$45.4	$76.8

Total Est. Annual Requirements

1975	$ 7.0
1980	$10.0

Source: Salomon Brothers' estimates.

munication in the United States. The concept of providing a fully integrated family of common user and private line message switching services took a giant step forward with the cutover in January of this year of the first of its Phase II ISCS Centers at Middletown, Virginia. This center, with a capacity of over one million prime time business messages per year, is currently being used to provide a wide variety of data communications services. Based on projections of growth for these varied services, revenues are likely to grow from the current $400 million level to over $1.5 billion by 1980. Since Western Union's computer-enhanced communications services provide an inherently efficient utilization of transmission and message switching facilities, this revenue level can be supported with a gross plant investment of only an estimated $3 billion, with $1.1 billion of external financing required.

Domestic satellites

Domestic satellites represent another exciting potential source of facilities for the data communications user in 1980. Here, however, it is necessary to realize the type of data communications which lend themselves feasibly to Domsat carriage. Due to the inherent time delay involved in an approximately 45,000-mile earth station-to-earth station path, these facilities are not really suitable for interactive types of data communications. Bulk transfer of one-way information, similar to the type of electronic mail currently being offered by the joint Postal Service/Western Union Mailgram service, is ideal, and certainly must be viewed as technically and economically feasible. Based on a projection of three operational systems carrying a variety of communications services by 1980, potential revenues of $400 million with a gross plant investment in both satellite and earth station facilities of about $1.2 billion seems reasonable. Taking into account the high depreciation rates on the satellites, we estimate that $0.8 billion will have to be raised from the capital markets.

Total requirements

The sum total of our projections shows a demand for new external capital of over $71 billion which must be raised in the ten-year period from 1971 to 1980. Additionally, over $5 billion must be raised to refinance telephone company bonds that will be maturing in the same period. The total capital requirements of the communications industry, therefore, exceeds $76 billion.

TABLE III—Capital Market Overview
(Billions of Dollars)

	1960	*1965*	*1970*	*1975E*	*1980E*
GNP	$504	$632	$977	$1300-1450	$1800-2200
Total Credit & Equity Market Supply	35	68	92	120-140	150-200

Source: Salomon Brothers' estimates.

This amounts to an annual rate of approximately $7 billion a year in 1975 and $10 billion in 1980. As shown on Table II, the telephone companies will use about 90 percent of this capital.

CAPITAL MARKETS THROUGH 1980

To fully understand the significance of these demands by the communications industry on the capital market, it is necessary to examine trends in both total money supply and demand in competing sectors of the economy.

GNP and net new capital

The GNP in current dollar terms stood at $504 billion in 1960, $632 billion in 1965, and $977 billion in 1970. This represents a growth in current dollar terms of over 6.8 percent per year. Looking out to 1975 and 1980 real growth in constant dollar terms of between 3.5 percent to 4.0 percent is estimated, with inflation of 3.0 percent to 4.5 percent yearly, yielding a growth in current dollar terms of 6.5 percent to 8.5 percent. It is estimated that the 1975 GNP will fall in a range of $1300 billion to $1450 billion, and the 1980 GNP will be between $1800 billion and $2200 billion (Table III).

In the late 1960s the average percent of the GNP saved and invested by government, business, and individuals ranged between 8.1 percent and 9.9 percent, with 9.0 percent as an average. Using the late 1960s as a representative model of what is likely to be encountered in the 1971-1980 time period, the total supply of debt and equity available for funding growth in the U.S. economy will likely be between $120 billion and $140 billion in 1975, and between $150 billion and $200 billion in 1980.

TABLE IV—Selected Demands Segments

	Percent		
	1961-65	*1966-70*	*1971E*
Private Mortgages	41.6	26.3	32.3
Total Corporate	12.4	22.7	23.5
State & Local	10.8	14.1	16.5
Federal Agencies	2.5	7.7	2.6
Sub-total	67.3	70.8	74.9
All Others	32.7	29.2	25.1

Source: Salomon Brothers' estimates.

Market demand

The total demand for funds can be categorized as follows:

A. Debt

1. Privately held mortgages
2. Corporate bonds
3. State and local securities
4. Foreign bonds
5. Business, consumer, and bank loans
6. Open market paper
7. Privately held Treasury and Federal Agency Securities

B. Corporate Equity

1. New cash offerings
2. Sale of stock options
3. Conversions of bonds

Of these categories, total corporate demand, mortgages, and certain governmental segments are of particular interest. The historic trends for these selected demand segments are shown in Table IV.

In the early 1960s corporations were relatively modest demanders of capital. In the 1961-65 period, corporate demand represented $35 billion, or 12 percent of the total demand for capital. By contrast, privately held mortgages consumed $117 billion, or 42 percent, of the total available funds. In the 1966-70 timeframe a number of factors caused significant changes. The credit crunch of 1966 coupled with the many statutory limitations on allowable interest that could be charged for mortgages caused the mortgage markets to tighten, and this sector declined to $105 billion, or 26 percent of total available funds. Corporate demands, on the other hand, more than doubled to a total of $91 billion, or 23 percent of total funds available. In 1971, with the "Federalization" of the mortgage market through such Federal agencies as FNMA, GNMA, and the Farm Home Loan Agency, mortgage demand is estimated at $40 billion, or 32 percent of totally available funds. Corporate demands, used largely to fund short-term liabilities, increased during the recent downturn in the economy, remaining relatively high at $29 billion or 23 percent.

State and local governments, responding to increasing pressure to meet the needs of society, have almost doubled their requirements for funds. In the 1961-65 period demand from this sector totaled $30 billion or 11 percent of total supply, while in the 1966-70 timeframe the comparable figures were $56 billion and 14 percent. For 1971, state and local governments required about $20 billion, or 16 percent, and demand from this sector is expected to continue to increase.

Another factor to consider is the growth in Federal Agency financing. In the 1961-65 timeframe these agencies funded $7 billion of debt—only 3 percent of the funds available. By the 1966-70 period total Agency demand had increased over four times to $31 billion or almost 8 percent of the total. The 1971 Agency demand is estimated at only $3 billion, or 3 percent.

CONCLUSIONS

Financial competition

Based on the above analysis, projected communications demands can now be placed in perspective, comparing them to the rising demand for corporate funds as a whole. In the 1961-65 period the communications industry required $7 billion in outside financing or some 20 percent of all corporate demand. Comparable figures for the 1966-70 timeframe were $13 billion and 14 percent. In both 1970 and 1971, however, annual requirements exceeded $5 billion and accounted for 17 -18 percent.

TABLE V—Communications Financing Demand
(Billions of Dollars)

	1961-65	*1966-70*	*1971*	*1971-75*	*1976-80*
Total Communications	$7.0	$12.7	$5.4	$31.2	$45.4
Percent Total Corporate	20	14	18	15-18	15-20

Source: Salomon Brothers' estimates.

As shown in Table V, it is estimated that the communications industry will continue to require 15-20 percent of the total corporate needs for external financing. In the coming decade, with increasing demands from all sectors of the economy, the industry will have to make itself relatively more attractive than its competitors in the capital markets in order to continue to attract this large a share. This means good balance sheets, "clean" accounting practices, attractive rates of return on both total capital and equity, and, of course, growth in earnings per share.

Very obviously, these requirements may be very difficult to achieve for the new entrants in the communications industry. In the more competitive, less sheltered atmosphere envisioned by both the FCC and the President's Office of Telecommunications Policy, a license to get into the business may not necessarily be a guarantee for making a profit, nor adequate collateral for obtaining capital funds. We feel very strongly, therefore, that communications in this decade will be shaped more decisively by financial constraints than by any other factor.

Data communications in 1980

In light of our view of the future of the communications industry as a whole, the following predictions can be made about the availability of data communications facilities in 1980.

A. The Bell System, with its continuing strong financial position, will continue to dominate the data communications business. Now that Bell has recognized that there will be a $5 billion data communications market in 1980, and that competition is increasing, we think you will see many more services geared to the data user. It is important to note that these services can be provided as an adjunct to voice (and, eventually, Picturephone services) with a relatively small incremental investment. Last year's announcement of future digital services, based on a "Data Under Voice" technique, is an indication of this approach. We also believe that, with the possible congestion of microwave frequencies for use in urban distribution of data communications signals, Bell will open up its cable distribution plant to low level, direct current type signals, eliminating the need for costly modems for limited distance use.

B. CATV will not have achieved its potential as a major provider of local data interchange services by 1980. The CATV business is a particularly capital-intensive one, whose growth will be financially limited. It is our opinion that the primary industry thrust in this decade will be to use its available financial resources to expand its subscriber base in the residential entertainment market. While certain token two-way services may be offered to serve specific markets or meet FCC regulations, full-scale development in this area will be postponed until the late Seventies or early Eighties.

C. The competitive response of the established common carriers to the MCI-type carriers will sufficiently limit the financial attractiveness of new point-to-point services, and keep them from becoming a major communications factor. DATRAN's nationwide, switched, digital system, on the other hand, is both technically unique and inherently more economical than either private line or switched analog facilit'es, or private line digital facilities. We feel DATRAN can be providing about 10 percent of the data communications traffic by 1980, assuming that its initially high capital requirements can be met.

D. We believe Western Union's centralized thrust—that of providing a fully integrated computer-centered message switching and transmission system to serve the record communications needs of business, government, and the general public—is an extremely viable one. Western Union's overbuild of its existing microwave network with full digital transmission capability very obviously allows them to make the most efficient use of their plant, with minimal capital investment. It is our feeling that Western Union will continue to aggressively market new communications services, as they have done with DATACOM, and that with the availability of their new Electronic Data Switch Network the company can be providing a whole range of switched and private line digital data services by 1975.

E. The use of satellite systems, except those provided by Bell and Western Union as fully integrated parts of their already existing and currently planned network operations, will be far too costly and far too limited for broad applicability to data communications.

The growth of data communications services and, in a broader sense, of the entire communications industry, is governed by the technological, marketing, and financial capabilities of the competing companies. Too often, discussions on the future of data communications concentrate on its exciting technological developments and potential applications demands while neglecting the very obvious fact that facility growth must be financed. As shown by the projections made in this paper, financing communications growth in the Seventies is a non-trivial problem with broad implications for users and providers alike.

Allocation of copies of a file in an information network

by R. G. CASEY

IBM Research Laboratory
San Jose, California

INTRODUCTION

We consider a mathematical model of an information network of n nodes, some of which contain copies of a given data file. Within this network, every node is able to communicate with every other node over communication links (a process which may entail routing through intermediate nodes). In particular, we are concerned with transactions with the multiply-located file. Such transactions fall into one of two classes: (1) query traffic between a node and the file, and (2) update traffic. An update message is assumed to be transmitted to every copy of the file, whereas a query is communicated only to a single copy.

We proceed to demonstrate, within a simple linear cost model for the network, several properties of the optimal assignment of files to nodes. One set of results expresses bounds on the number of copies of the file that should be included in the network, as a function of the relative volume of query and update traffic. Secondly, a test useful in determining the optimum configuration is derived.

The problem of allocating resources in a network first arose within the context of determining the most economical location for manufacturing plants and warehouses.[1,2,3] In some studies, the model of the network is made in such a way that an efficient algorithm yielding the optimal allocation is not readily obtainable, and heuristic or approximate techniques are tried instead. Frazer,[4] for example, has developed a method for allocating production facilities when the cost of deploying a resource at a given node consists of a fixed overhead expenditure plus an amount proportional to the total demand on the facility.

These manufacturing models are conceptually different from the file allocation problem, when updating traffic is considered in the latter. Since all copies of a file are modified by each update message, the total volume of data transmitted in the network is not independent of the allocation policy, as is the total volume of goods shipped in a manufacturing environment, but rather increases with the number of file copies allocated. Yet, as will be shown, the models are mathematically equivalent to the file model analyzed here.

Chu[5] has investigated a linear programming solution to the file allocation problem. His model includes storage costs and queuing delays, but the number of copies of each file in the system is assumed to be known. Whitney[6] has formulated a similar model, and applied it to the task of designing network topology, as well as that of allocating copies of the file.

THE MODEL

The fixed cost (mainly for storage) of locating a copy of the file at the kth node ($k=1, 2, \ldots, n$) will be denoted by an amount σ_k, measured in, say, dollars per month. The symbols λ_j and ψ_j will be used to represent the volume of query traffic and of update traffic, respectively, emanating from node j ($j=1, 2, \ldots, n$). The quantities d_{jk} and d_{jk}' are the costs of a unit of communication from node j to node k for a query and an update transaction, respectively. Thus, d_{jk} and d_{jk}' might be measured in dollars per megabit transmitted, and λ_j and ψ_j expressed in megabits per month. The possibility of a difference in cost rates is included in the model on the premise that in many applications updates can be accumulated and transmitted either via a cheaper medium (e.g., by mailing magnetic tapes), or at a time when communications rates are lower (e.g., at night using switched lines).* Queries are assumed to be entered on-line, at a higher communication rate. If the kth node contains a copy of the file, then communications from the jth node produce a cost term of $\lambda_j d_{jk}$ for query transactions and an amount $\psi_j d_{jk}'$ for update

* The author is indebted to Dr. W. D. Frazer of IBM Research for pointing out this generalization.

transactions. Actually, in much of the discussion that follows, and in the experiments carried out, we assume equal cost rates for inquiry and update.

In practice, the cost of shipping information may not be linear with the amount sent; the assumption is nonetheless of theoretical interest in order to obtain a first-order approximation to a rather complex monotonically increasing function.

We are concerned with the problem of determining at which nodes of the network copies of the file shall reside. We shall assume that this allocation is to be done in such a way as to minimize the total cost of communication between users and files. In general, the cost of querying is reduced as we increase the number of file nodes in the network (users near a new file node find their querying more economical; the other users are no worse off). On the other hand, storage costs and the cost of updating go up as new copies of the file are introduced, since every copy must be updated. In the limiting cases, if no updating is done, and if storage costs are low, a copy of the file should be kept at every node (and network communication not used), whereas if only updating is done, and no querying, a single copy of the file should be maintained at some optimal node of the network.

By an assignment of the file we shall mean a choice of nodes at which to locate the file. Let I denote a set of node indexes representing a given assignment. The total communication cost resulting from this choice of file locations is a sum over individual user nodes. In the case of query traffic, we assume that the user accesses that copy of the file which minimizes his communication cost. His updates, of course, must be transmitted to all the file nodes. The general expression for total cost is therefore:

$$C(I)=\sum_{j=1}^{n}\left[\sum_{k\epsilon I}\psi_j d_{jk}'+\lambda_j \min_{k\epsilon I} d_{jk}\right]+\sum_{k\epsilon I}\sigma_k$$

The problem of file allocation in this context is to choose the index set, I, so as to minimize $C(I)$.

This cost function can be written in the form

$$C(I)=\sum_{k\epsilon I} U_k+\sum_{j=1}^{n} G_j(I)$$

where

$$U_k=\sigma_k+\sum_{j=1}^{n}\psi_j d_{jk}'$$

$$G_j(I)=\lambda_j \min_{k\epsilon I} d_{jk}$$

With $C(I)$ expressed in this form the problem of minimizing cost is seen to be exactly the problem investigated by Efroymson and Ray,[1] Feldman, et al.,[2] Frazer,[4] and others. The update costs of the file model are analogous to the fixed costs of the plant location model, while query costs are analogous to the transportation costs. Previous researchers tried mixed integer-linear programming techniques for obtaining solutions to the minimization problem. Generally, the true minimum of $C(I)$ is computationally very expensive to obtain by such methods, and heuristics are developed that sacrifice optimality in order to arrive at "good" solutions quickly.

Here, we take an alternative approach. We shall first exhibit several mathematical properties of the cost function. We shall use these to construct a search procedure which is guaranteed to produce the minimum of $C(I)$. Finally, we suggest heuristic modifications to the general agorithm for use in cases where it is too expensive to be applied. Thus, both attempts to solve the problem, by means of integer programming and by exploiting the properties of the cost function, turn to *ad hoc* adjustments in order to obtain relief from the expense of guaranteeing an optimum solution. It would be of interest to compare the performance of the two approaches on the same problems.

The initial examination of the properties of $C(I)$ yields bounds on the number of elements in the optimal file node set I, when storage costs do not vary. This will be shown by means of the following theorem.

Let the cost of update communication be equal to the cost of query communication ($d_{jk}=d_{jk}'$).

Theorem 1:

If for some integer $r\leq n$ $\psi_j>\lambda_j/(r-1)$ for each $j=1, 2, \ldots, n$ then any r-node file assignment is more costly than the optimal one-node assignment.

We prove the theorem by means of the following lemma:

Lemma:

If $\psi_j=\rho\lambda_j$ for $j=1, 2, \ldots, n$ then, an r-node assignment cannot be less costly than the optimal one-node assignment if $\rho\geq 1/(r-1)$.

Proof of Lemma:

Consider an arbitrary assignment $I=\{1, 2, \ldots, r\}$. Let the elements of I be ordered such that node 1 is the lowest cost single-node assignment.

We have

$$C(I)=\rho\sum_{k=1}^{r}\sum_{j=1}^{n}\lambda_j d_{jk}+\sum_{j=1}^{n}\min_{k\epsilon I}\lambda_j d_{jk}+\sum_{k=1}^{r}\sigma_k$$

and

$$C(\{k\})=(1+\rho)\sum_{j=1}^{n}\lambda_j d_{jk}+\sigma_k \qquad k=1,2,\ldots,r$$

By the optimality of node 1 there are nonnegative numbers $\alpha_2,\ldots,\alpha_r$ such that

$$\sum_{j=1}^{n}\lambda_j d_{jk}=\sum_{j=1}^{n}\lambda_j d_{j1}+\alpha_k+\frac{\sigma_1-\sigma_k}{1+\rho} \qquad k=2,3,\ldots,r.$$

Substituting the above, we can write

$$C(I)-C(\{1\})=[\rho r-\rho-1]\sum_{j=1}^{n}\lambda_j d_{j1}+\sum_{k=2}^{r}\rho\alpha_k$$
$$+\sum_{j=1}^{n}\min_{k}\lambda_j d_{jk}+\frac{\rho(r-1)\sigma_1}{1+\rho}$$
$$+\frac{1}{1+\rho}\sum_{k=2}^{r}\sigma_k$$

which is certainly nonnegative if $(\rho r-\rho-1)$ is nonnegative. That is, $\rho\geq 1/(r-1)$ implies that $C(I)\geq C(\{1\})$.

If the optimal single node assignment is in I, then it is node 1 and the lemma is proved. If the optimal node, say k', is not in I, then we have $\rho\geq 1/(r-1)$ implies $C(I)\geq C(\{1\})>C(\{k'\})$ and so the lemma is true in this case as well.

Proof of Theorem 1

The lemma concerns the case where each user generates the same proportion of update traffic to query traffic. Suppose now that the proportions vary from node to node, but always exceed a given amount ρ. Then, there are nonnegative quantities ϵ_j such that

$$\psi_j=(\rho+\epsilon_j)\lambda_j \qquad j=1,2,\ldots,n$$

The cost functions can now be written:

$$C'(\{1\})=\sum_{j=1}^{n}(1+\rho+\epsilon_j)\lambda_j d_{j1}+\sigma=C(\{1\})+\sum_{j=1}^{n}\epsilon_j\lambda_j d_{j1}$$

$$C'(I)=\sum_{j=1}^{n}(\rho+\epsilon_j)\lambda_j\sum_{k=1}^{r}d_{jk}$$
$$+\sum_{j=1}^{n}\min_{k}(\rho+\epsilon_j)\,d_{jk}+r\cdot\sigma$$
$$=C(I)+\sum_{j=1}^{n}\epsilon_j\lambda_j\sum_{k=1}^{r}d_{jk}+\sum_{j=1}^{n}\epsilon_j\min_{k}d_{jk}$$

where the unprimed costs are those given in the lemma.

But clearly $C'(I)\geq C'(\{1\})$ whenever $C(I)\geq C(\{1\})$. Then applying the lemma, the condition $\rho\geq 1/(r-1)$ is sufficient to ensure that the r-node assignment is not optimal. In addition, if at least one of the ϵ_j's is greater than zero, we must have the strict inequality $C(I)>C(\{1\})$.

Corollary 1

If the update/query traffic ratio satisfies $\rho\geq 1/(r-1)$ (r an integer) then the optimal allocation consists of no more than r nodes.

Proof

If $\psi_j\geq\lambda_j/(r-1)$ for each j then certainly for any integer l

$$\psi_j>\lambda_j/(r+l-1)$$

and so theorem 1 rules out the optimality of an $(r+l)$-node assignment.

Corollary 2

If each user generates at least 50 percent of his traffic in the form of updates, then the optimal assignment policy is to locate only a single copy of the file in the network.

The corollary follows directly from the theorem by setting $r=2$, but is worth stating separately since it sets a bound beyond which multiple copies of the file should not be considered. Furthermore, it is easy to show that for equal proportions of update and query traffic the two-node assignment is no more costly than a one-node assignment of the file only if storage costs are neglected, and the rows and columns of the cost matrix can be permuted to yield the form:

$$\begin{bmatrix} 0 & 0\ldots 0 & a_1 & a_2\ldots a_m & 0 & 0\ldots 0 \\ b_1 & b_2\ldots b_l & 0 & 0\ldots 0 & 0 & 0\ldots 0 \\ \multicolumn{6}{c}{\text{(rest of matrix)}} \end{bmatrix}$$

where

$$\sum_{i=1}^{m}a_i\lambda_a=\sum_{i=1}^{m}b_i\lambda_b,$$

λ_a and λ_b being the respective traffic volumes for the first two rows (query or update).

A PROPERTY OF THE OPTIMAL FILE ALLOCATION

In this section, we examine the behavior of the cost function as additional file nodes are added. In particular, we consider a graph such as Figure 1, where each vertex is identified with a file assignment (denoted by a binary vector having 1's in those positions corresponding to file nodes, 0's elsewhere). Associated with each vertex is the corresponding value of the cost function (not shown in Figure 1). For mathematical convenience the null vertex is assigned infinite cost. The edges of the graph are directed paths corresponding to the addition of a single file node to the previous assignment. The graph is conveniently arranged in levels where the vertices at the kth level represent all k-node file assignments.

We demonstrate the significant property of this graph (which we shall call the "cost" graph) that if a given vertex has a cost less than the cost of any vertex along the paths leading to it, then the sequence of costs encountered along any one of these paths decreases monotonically. Thus, in order to find the optimal allocation policy, it is sufficient to follow every path of the cost graph until the cost increases, and no further. Because of this property, only a small subset of the possible assignments need be tested, compared with the exhaustive search procedure in which 2^n different file allocations must be evaluated.

The monotonicity will be exhibited in two stages: first, for the case of two steps through the graph, and then for the general case.

Let $I\sim X$, where $X\subset I$, denote the index set I with the elements of set X removed.

Consider an arbitrary file assignment corresponding to index set I, and assume the nodes are so numbered that $I=[1, 2, \ldots, r]$. We proceed to show:

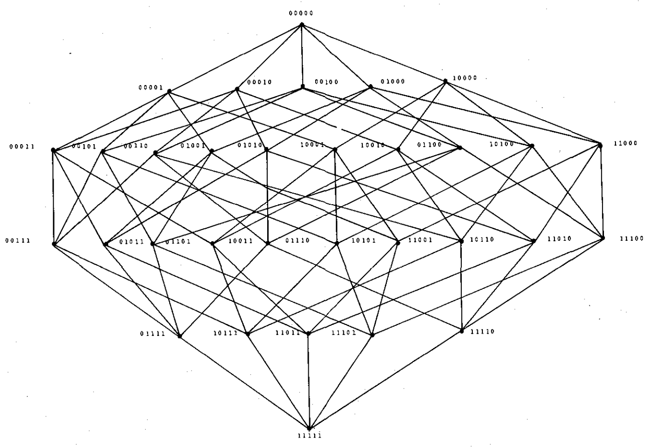


Figure 1—Graph of the allocation process

Lemma:

If $C(I)\leq C(\mathrm{I}\sim\{k\})$ for $k=1, 2$, then $C(I\sim\{k\}\leq C(I\sim\{1, 2\})$ for $k=1, 2$.

Proof of Lemma:

The cost function can be written, in general,

$$C(I\sim X)=\sum_{k\epsilon I\sim X} U_k+\sum_{j=1}^{n}\lambda_j \min_{k\epsilon I\sim X} d_{jk}$$

Thus

$$C(I)-C(I\sim\{1\})=U_1-\sum_{j=1}^{n}\lambda_j\Delta_j$$

where

$$\Delta_j=\min_{k\epsilon I\sim\{1\}} d_{jk}-\min_{k\epsilon I} d_{jk}$$

Also

$$C(I\sim\{2\})-C(I\sim\{1, 2\})=U_1-\sum_{j=1}^{n}\lambda_j\Delta_j'$$

where

$$\Delta_j'=\min_{k\epsilon I\sim\{1,2\}} d_{jk}-\min_{k\epsilon I\sim\{2\}} d_{jk}$$

Consider the difference

$$\Delta_j'-\Delta_j=\min_{k\epsilon I\sim\{1,2\}} d_{jk}+\min_{k\epsilon I} d_{jk}-\min_{k\epsilon I\sim\{2\}} d_{jk}-\min_{k\epsilon I\sim\{1\}} d_{jk}$$

We have

$$\min_{k\epsilon I} d_{jk}=\min(\min_{k\epsilon I\sim\{1\}} d_{jk}, \min_{k\epsilon I\sim\{2\}} d_{jk})$$

$$\min_{k\epsilon I\sim\{1,2\}} d_{jk}\geq\max(\min_{k\epsilon I\sim\{1\}} d_{jk}, \min_{k\epsilon I\sim\{2\}} d_{jk})$$

Therefore

$$\Delta_j'-\Delta_j\geq 0, \text{ that is } \Delta_j'\geq\Delta_j,$$

and so

$$C(I\sim\{2\})-C(I\sim\{1, 2\})\leq C(I)-C(I\sim\{1\})$$

thus,

$$C(I)\leq C(I\sim\{1\})$$

implies that

$$C(I\sim\{2\})\leq C(I\sim\{1, 2\})$$

By a mere permutation of indexes we may likewise prove that

$$C(I)\leq C(I\sim\{2\})$$

implies that

$$C(I\sim\{1\})\leq C(I\sim\{1,2\})$$

and so the lemma is true.

By means of this lemma, we can now prove the general theorem:

Theorem 2:

Given an index set $X\subseteq I$ containing r elements, and having the property

$$C(I)\leq C(I\sim\{x\}) \text{ for each } x\epsilon X$$

then for every sequence $R^{(1)}, R^{(2)}, \ldots, R^{(r)}$ of subsets of X such that $R^{(k)}$ has k elements, and $R^{(k)}\subset \mathrm{R}^{(k+1)}$ it is true that

$$C(I)\leq C(I\sim R^{(1)})\leq C(I\sim R^{(2)})$$
$$\leq C(I\sim R^{(3)})\leq \cdots \leq C(I\sim R^{(r)})$$

Proof: (by induction)

The first inequality above is given in the hypothesis. Since $R^{(2)}$ has two elements the second inequality follows from the lemma. It may thus be taken as hypothesis to prove the third, and so on. Each inequality proved can be used together with the lemma to prove the inequality to its right.

Observations:

If we take $X=\mathrm{I}$ in the theorem, then it states that along any path in the cost graph from the null vertex (which we have assigned infinite cost) to the vertex corresponding to I, cost decreases monotonically from one vertex to the next, providing of course that index set I satisfies the hypothesis of the theorem. The monotonic decreasing property can also be shown to hold in the reverse direction; i.e., for paths from the vertex $\{1, 2, \ldots, n\}$ to the optimum.*

APPLICATION TO FILE ALLOCATION

The theorem just proved provides a test that is useful in determining the minimum cost allocation of file copies to nodes of the network. Referring to the cost graph (Figure 1), let us define an "antecedent" of an arbitrary vertex v as a vertex which has a connection to v from the next lower level, and a "successor" of v as a vertex connected to v at the next higher level. An antecedent contains the file nodes of v less one node, while each successor contains the nodes of v plus one additional node. A vertex at the rth level of the graph has r antecedents, and $(n-r)$ successors, where n is the number of nodes in the network. (Note that, for clarity, we use the term "node" in referring to the computer network, and the term "vertex" with respect to the cost graph.)

We shall also define a "local optimum" of the cost graph as a vertex which is less costly than all of its antecedents and successors. Clearly, the global optimum we seek belongs to the set of local optima of the graph.

The theorem permits us to discover all the local optima without computing the cost of every vertex, for each path leading from the 0 level to a local optimum must give rise to a monotonically decreasing sequence cf costs. Whenever an increase is encountered in a step forward through the graph, that path can be abandoned since it cannot lead to a local (or global) optimum. A "path-tracing" routine which evaluates each possible sequence of node additions in this way is certain to produce the minimum value of $C(I)$. The amount of computation needed to extract the minimum will increase with the number of local optima, and with the number of nodes in the optimal configuration.

A computer algorithm can be implemented in several different ways to select file nodes one at a time up to the optimum configuration. One approach is to follow all paths in parallel through the cost graph, stepping

INPUT PARAMETERS

COST PER MEGABYTE SHIPPED

		QUERY	UPDATE	FILE	NODES			
		TRAFFIC	TRAFFIC	1	2	3	4	5
U	1	24	2	0	6	12	9	6
S	2	24	3	6	0	6	12	9
E	3	24	4	12	6	0	6	12
R	4	24	6	9	12	6	0	6
S	5	24	8	6	9	12	6	0

QUERYING COSTS

	1	2	3	4	5
1	0	144	288	216	144
2	144	0	144	288	216
3	288	144	0	144	288
4	216	288	144	0	144
5	144	216	288	144	0

UPDATE COSTS

	1	2	3	4	5
1	0	12	24	18	12
2	18	0	18	36	27
3	48	24	0	24	48
4	54	72	36	0	36
5	48	72	96	48	0

Figure 2—A five-node example

* A reviewer pointed out this generalization.

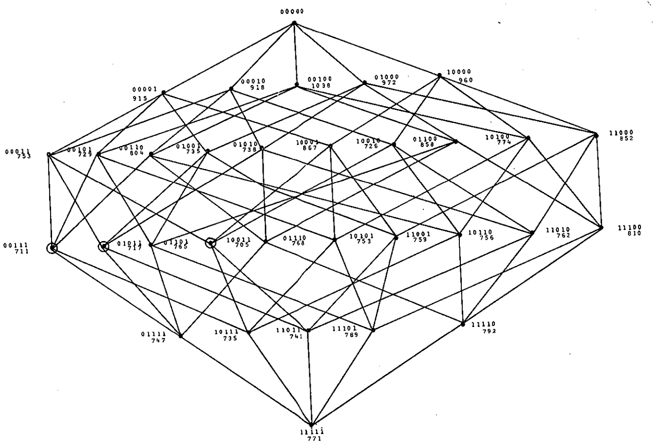


Figure 3—The cost graph for the example. Local optima are circled

one level per iteration. This method is computationally efficient, but may require a great deal of storage in a large problem. Alternatively, a program can be written to trace systematically one path at a time. Such a technique uses less storage, but may require redundant calculations since many different paths intersect each vertex.

EXAMPLES

We consider a five-node network with query and update parameters, and cost matrix as in Figure 2. Figure 3 shows the file node cost graph, and indicates the optimal allocations. This small-scale example illustrates the monotonicity of the cost function along paths leading to a local optimum.

Note that the matrix $\{d_{jk}\}$ is symmetric and has zero elements along the main diagonal. This is a plausible condition in a practical case, but is not required in any way by the theorems proven here.

As a second example, we postulate the ARPA computer network and its traffic matrix as given by Kleinrock.[7] In order to treat this case, which is not initially posed as a centralized data base configuration, we make the following rather arbitrary assumptions:

(1) All the resources of the network represented in the traffic matrix are gathered into one large file, copies of which are to be allocated to a subset of the nodes.

(2) The query traffic (Figure 4) from each user to the file is as given under the "node output" column in Kleinrock's table. If the user is accessing a program, we may think of this "query" traffic as comprising the command messages he must send in order to run the program, plus the program output sent back to him. (Note: Kleinrock does not give figures for each node's use of its own facilities.)

(3) "Update" traffic, which causes modification of programs and data in all copies of the file, is a fixed percentage of the query traffic defined in (2). The update/query ratio is the same for each user, and is varied as a parameter in the computer runs described below.

(4) The "cost" of communicating between two network nodes is as shown in Figure 4. The quantities given here are roughly the straight-line distances between nodes. Thus, the cost function to be minimized is the total amount of "flow" in the network, i.e., the product of message length in bits, multiplied by the distance through which the message must travel on a direct path to its destination, summed over all messages. In this view an update message is assumed to be sent separately to each file location, rather than relayed.

Assumptions (1)-(4) ignore the network communication links as configured by Kleinrock. Ordinarily these would determine the cost matrix. Substituting "distance" for "cost," as done here, might be expedient in a preliminary analysis to allocate resources prior to a topological design of the network. We employ the ARPA data primarily because it tests our model on a problem involving many nodes. In addition, we should like to encourage speculation on the concept of multiple copies of a resource (data or programs), and whether such a facility is attractive in future ARPA-like networks.

Figure 5 summarizes the file allocation experiments conducted using this data, in which the proportion of update traffic to query traffic was varied from 100 percent to 10 percent in steps. As expected, only single node allocations are indicated when the two types of traffic are equal. As a smaller volume of update traffic is generated, multiple node solutions begin to appear. The copies of the file are widely distributed geographically, and the single node solution is centrally located. However, such features result from the particular traffic distribution assumed, and are not generalizations valid for every case.

It is interesting to note the large number of local optima generated (see Figure 5). Because certain ARPA sites, e.g., MIT, BBN, Harvard, and Lincoln Laboratories, are very near one another, if one occurs in a local optimum one of its neighbors may be substituted for it to produce a second configuration that

COST PER MEGABYTE SHIPPED

		QUERY	FILE	NODES																	
		TRAFFIC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
U	1	8	0	75	75	75	75	350	450	490	640	900	1050	2080	2700	2700	2700	2670	2610	2610	2610
S	2	21	75	0	5	5	5	300	400	480	660	920	1060	2110	2720	2720	2720	2700	2640	2640	2640
E	3	7	75	5	0	5	5	300	400	480	660	920	1060	2110	2720	2720	2720	2700	2640	2640	2640
R	4	6	75	5	5	0	5	300	400	480	660	920	1060	2110	2720	2720	2720	2700	2640	2640	2640
S	5	10	75	5	5	5	0	300	400	480	660	920	1060	2110	2720	2720	2720	2700	2640	2640	2640
	6	4	350	300	300	300	300	0	160	280	500	710	850	1940	2570	2570	2570	2520	2450	2450	2450
	7	3	450	400	400	400	400	160	0	190	430	610	730	1850	247	247	247	240	233′	233	233
	8	10	490	480	480	480	480	280	190	0	240	430	570	1670	2300	2300	2300	2240	2170	2170	2170
	9	9	640	660	660	660	660	500	430	240	0	280	430	1460	2090	2090	2090	2050	1980	1980	1980
	10	50	900	920	920	920	920	710	610	430	280	0	160	1250	1860	1860	1860	1800	1730	1730	1730
	11	3	1050	1060	1060	1060	1060	850	730	570	430	160	0	1160	1760	1760	1760	1680	1610	1610	1610
	12	26	2080	2110	2110	2110	2110	1940	1850	1670	1460	1250	1160	0	610	620	620	640	600	600	600
	13	20	2700	2720	2720	2720	2720	2570	2470	2300	2090	1860	1760	610	0	40	40	290	340	340	340
	14	21	2700	2720	2720	2720	2720	2570	2470	2300	2090	1860	1760	620	40	0	5	250	320	320	320
	15	3	2700	2720	2720	2720	2720	2570	2470	2300	2090	1860	1760	620	40	5	0	250	320	320	320
	16	5	2670	2700	2700	2700	2700	2520	2400	2240	2050	1800	1680	640	290	250	250	0	80	80	80
	17	4	2610	2640	2640	2640	2640	2450	2330	2170	1980	1730	1610	600	340	320	320	80	0	10	10
	18	8	2610	2640	2640	2640	2640	2450	2330	2170	1980	1730	1610	600	340	320	320	80	10	0	5
	19	7	2610	2640	2640	2640	2640	2450	2330	2170	1980	1730	1610	600	340	320	320	80	10	5	0

Figure 4—The ARPA example. The locations of the nodes (from Kleinrock[7]) are: (1) Hanover, N.H., (2)-(5) Boston Area, (6) Murray Hill, N.J., (7) Washington, D.C., (8) Pittsburgh, Pa., (9) Ann Arbor, Michigan, (10) Urbana, Illinois, (11) St. Louis, Mo., (12) Salt Lake City, Utah, (13)-(15) San Francisco Bay Area, (16) Santa Barbara, California, (17)-(19) Los Angeles Area

is also locally optimum. Most of the local optima are due to this phenomenom, suggesting that greater computational efficiency would have resulted from lumping neighboring sites into a single network node for an initial application of the algorithm, followed by a stage of finer calculations. Such artifices are hardly necessary in problems of this size, however. The sequence of six runs depicted in Figure 5 was carried out on an IBM 360/91 in less than ten seconds, including Fortran compilation.

EXTENSIONS

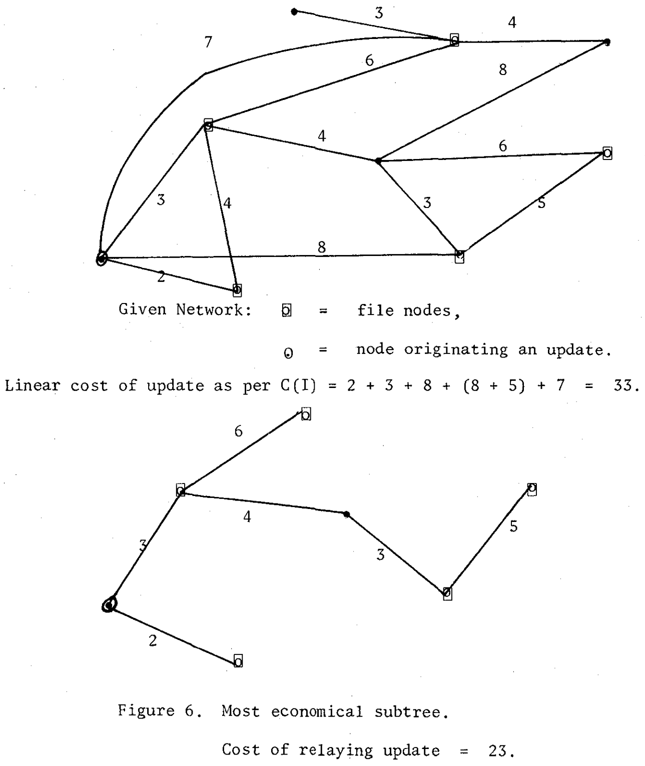
Path tracing on a related problem

One feature of the path-tracing technique is its versatility. It can be applied to problems that do not fit the linear programming framework. As one example consider the following modification of the file allocation model. The cost function as defined in the text assumes that the expense of transmitting an update message to all copies of the file is the sum of the costs of communicating with each copy separately, as if a distinct message had to be sent from the updater to every file node.

Update/Query Percent	Optimal Allocation	Cost	#Local Minima
10	2, 10, 14	117, 544	140
20	9, 14	188, 738	88
30	10, 14	242, 546	88
40	10, 12	291, 754	77
100	10	427, 460	19

Figure 5—Results for the ARPA example

Actually, the preferred mode of operation would be to relay the message through the network in the most economical manner. The resulting cost is less than that incurred by sending duplicate messages. It is the cost associated with the most economical subtree of the network that contains the originating node and the file nodes (see Figure 6). This new cost function defined when update costs are calculated in this way is nonlinear in a very complex way. Yet the path-tracing algorithm can still be applied to the allocation problem. We nò longer have the assurance provided by Theorem 2, which guarantees that the method will generate the optimal allocation. However, the cost function behaves in much the same manner as before. It still consists of two parts: an update (and fixed cost) term that increases as additional file copies are allocated, and a query term that decreases with additional copies. The algorithm may be heuristic in the new framework, but with only slight modification it can be programmed to determine at least allocations of the type we have called "local optima," of which the true optimum is one.



Given Network: ⊡ = file nodes,

⊙ = node originating an update.

Linear cost of update as per C(I) = 2 + 3 + 8 + (8 + 5) + 7 = 33.

Figure 6. Most economical subtree.

Cost of relaying update = 23.

Figure 6—Most economical subtree for relaying an update message

Heuristic modification of the path-tracing routine

If the optimal solution to the allocation problem consists of many network nodes then the general rule of following every monotonically decreasing path in the cost graph may be computationally too expensive. For example, if the optimum contains 40 nodes then at least $2^{40} \doteq 10^{12}$ values of the cost function must be evaluated. In order to decrease the amount of computation, it is necessary to sacrifice optimality for speed. We suggest several methods by which this may be done in a reasonable manner.

Assume that the cost graph is being searched level-by-level (i.e., in parallel). At each level, the vertices which are lower in cost than their antecedents are recorded. We call these vertices "admissible" and denote them by the set A_k at the kth level. We wish to carry only a limited number of the members of A_k into the $(k+1)$th stage. One plausible selection rule is to keep the vertices having the lowest cost. Alternatively, we may reason that if two assignments are similar (i.e., differ in only a few file nodes) then probably their union will also lie on a monotonic cost path, and it would be redundant to trace paths through both. In this view, we want to select the most "dissimilar" vertices from A_k. Several techniques are available to do this selection. One method, previously used in pattern recognition, is the following. Order the members of A_k arbitrarily, specify a threshold τ (a parameter), and examine the members of A_k in order. Denote by A_k' the subset being formed. The first member of A_k' is the first member of A_k, say a_1. The next member of A_k' (denoted a_2) is the first member of A_k which differs from a_1 by at least τ units. Next, test for an a_3 which differs from both a_1 and a_2 by at least τ and so on. If A_k' has too many members, or too few, the routine may be repeated with τ larger, or smaller, respectively. The set A_k' is used to generate A_{k+1}, which is in turn thinned down to A_{k+1}'.

A third method has actually been programmed and compared with the general search technique. The procedure is to do complete path-tracing (applying Theorem 2) up to a level of the cost graph at which set A_k begins to get too large. From this level on, only the most "promising" paths are followed. For example, as programmed the cost graph is completely evaluated through the second level (2-file nodes per vertex). From each admissible second level vertex only a single path is followed; namely, that path which gives minimal cost at each succeeding level. Since this program only has one-step look ahead, the optimum may be missed. In sample runs on the ARPA problem, the optimum was always found (for each value of the update/query parameter), although local optima were sometimes overlooked.

This heuristic program was also tested on the ARPA configuration using the revised cost function described above, i.e., a function which assumes that updates are relayed between file notes in the most economical manner. Use of a simplified program is called for in this case; the heuristic program required 12 minutes to repeat the five runs of Figure 5, which were performed in under ten seconds using the linear cost relationship. Figure 7 shows the allocations produced by this experiment. These assignments may not be global optima. They do satisfy our definition of a local optimum. Note that, as expected, for the same traffic more copies of the

Update/Query Percent	Optimal Allocation (Nodes)	Cost
10	2, 6, 8, 9, 10, 12, 13, 14, 16, 18	38, 285
20	2, 8, 10, 12, 14	105, 800
30	2, 8, 10, 12, 14	171, 225
40	8, 10, 12	227, 340
100	10	427, 460

Figure 7—Allocation results using the nonlinear cost model (ARPA network)

file are assigned if updates are relayed. In addition, total cost is reduced markedly under conditions of low update traffic, indicating the importance of including the more complex cost relations in the model.

CONCLUSION

The analytical properties of a linear-cost model of an information network have been investigated. The proportions of update traffic to query traffic generated by the users of a given file in the network were shown to determine an upper bound on the number of copies of the file present in the least-cost network. In addition, a basic property of the cost function was demonstrated and shown to justify a path-tracing procedure for determining the optimal assignment of copies of the file to nodes of the network.

The model, while simple, expresses relevant features of the tradeoff between the costs of querying and updating in the network. Presumably, the general properties derived apply at least approximately when more complex models are considered.

ACKNOWLEDGMENT

Dr. M. E. Senko first pointed out to the author the need for an allocation model which would distinguish between the query and update activity in a network. The work reported here owes additional debts to the encouragement and commentary of Drs. C. P. Wang, H. Ling and V. Lum.

REFERENCES

1 M A EFROYMSON T L RAY
A branch-bound algorithm for plant location
Operations Research Vol 14 No 3 pp 361-368 May-June 1966

2 E FELDMAN et al
Warehouse location under continuous economies of scale
Management Science Vol 12 No 9 pp 670-684 May 1966

3 K SPIELBERG
Algorithm for the simple plant-location problem with some side conditions
Operations Research Vol 17 1969 pp 85-111

4 W D FRAZER
An approximate algorithm for plant location under piecewise linear concave costs
IBM Research Report RC 1875 IBM Watson Research Center Yorktown Heights N Y July 25 1967

5 W W CHU
Optimal file allocation in a multiple computer system
IEEE Trans on Computers Vol C-18 No 10 October 1969

6 V K M WHITNEY
A study of optimal file assignment and communication network configuration in remote-access computer message processing and communication systems
SEL Technical Report No 48 Systems Engrg La
Dept of Elect Engrg University of Michigan September 1970 (PhD Dissertation)

7 L KLEINROCK
Models for computer networks
Proc International Conference on Comm Boulder Colorado pp 2.9-2.16 June 1969

Production and utilization of computer manpower in U.S. higher education*

by JOHN W. HAMBLEN

Southern Regional Education Board
Atlanta, Georgia

SURVEYS

During the past five years the Computer Science Project of the Southern Regional Education Board (SREB) has conducted three surveys on Computers in U.S. Higher Education including their utilization and related educational programs. These studies have resulted in three publications[1,2,3] the first by the SREB and the latter two by the National Science Foundation.

The first survey was a stratified random sample of 739 institutions, from a population of 2219, selected by systematic random sampling within strata by the U.S. Office of Education. Six hundred sixty nine or 92 percent of the institutions in the sample responded.

The base year for data collection was 1964-65 and projections were requested from the institutions for 1968-69. The data collection instrument was designed by the Mathematical Sciences Section of NSF to meet their existing program needs for planning. Consequently, emphasis was placed upon the financial aspects of college and university computer center operations and only limited information was gathered with regard to manpower training and utilization.

The second survey was much more comprehensive and was sent to all institutions of higher education listed in the U.S. Office of Education's Higher Education General Information Survey (HEGIS) file. Nineteen hundred sixty five or 79 percent of the 2477 institutions responded. The data collection forms were designed with the advice and counsel of thirty or more national professional and institutional associations and government agencies. The base year for reporting data was 1966-67.

Extensive modifications were made in the data collection forms for the third survey. The base year was 1969-70.

Problems resulting from lack of standardization in the computer industry are well-known. Unfortunately the educational and occupational nomenclature also suffers this malady. The shifting of terms and definitions over the years can inject disastrous consequences on estimates and, particularly, projections. Attempts to make comparisons with other studies become even more hazardous. In the following only estimates and projections based upon the three surveys described above will be presented. Two other papers in these proceedings[4,5] by Gilchrist and Weber will treat comparisons and incorporate higher education's role in manpower production and utilization to give a total picture of the supply and demand for computer personnel in the U.S.

MANPOWER PRODUCTION

Educational programs in Computer Science, Data Processing, Information Science, etc., exist in institutions of higher education at all levels. Two year programs terminate with an associate degree and four year programs with a bachelor's degree. Master's and doctorate programs exist at many institutions. Although computer science and data processing are by far the most popular program names (see Table I) there appears still to be far too many different names used for educational programs in institutions of higher education. Further consolidation is necessary to ensure a healthy and unified development of the disciplines. The efforts of the ACM Curriculum Committee on Computer Science[6] and the ACM Committee on Computer Education for Management[8] have and will continue to have a unifying influence on academic programs in computer and information science. The work of these

* The data upon which this paper is based was collected and processed through partial support from the National Science Foundation under contracts C465, C508 and C604. The SREB Computer Sciences Project is also supported by grants from the Esso Education Foundation and the IBM Corporation.

TABLE I—Reported Degree Programs in Computer Science, Data Processing, Information Science, etc. for Academic Years 1966-67, 1969-70, 1970-71, and 1971-72

	Associate				Bachelor's				Master's				Doctorate			
Program Name	66-67	69-70	70-71	71-72	66-67	69-70	70-71	71-72	66-67	69-70	70-71	71-72	66-67	69-70	70-71	71-72
Data Processing: Total	122	204	217	219	15	21	25	26	5	6	5	5	1	1	2	1
In Departments of:																
Data Processing		136	143	143	11	5	6	6	4	1	1	1	1	0	0	0
Business Administration	0	34	36	38	1	11	12	13	1	2	1	1	0	0	0	0
Business		17	20	20		2	2	2		2	2	2		0	1	1
Computer Science		3	3	3		1	1	1		0	0	0		0	0	0
Miscellaneous (seven)[1]	0	14	15	15	3	2	4	4	0	1	1	1	0	1	1	0
Computer Science: Total	8	33	37	42	50	90	113	133	58	76	86	102	35	50	54	59
In Departments of:																
Computer Science	7	20	23	27	30	40	52	66	35	37	39	49	22	24	25	28
Mathematics	0	0	0	0	7	19	23	26	4	8	8	9	4	3	3	3
Electrical Engineering	0	0	0	0	8	8	10	11	9	12	13	14	6	9	10	10
Computer and Information Science		1	1	1		1	3	3		3	3	3		2	2	2
Information and Computer Science		0	0	0		2	2	2		1	1	2		2	2	3
Management Science	0	2	2	3	0	6	7	8	3	0	1	2	1	0	0	0
Industrial Engineering	0	0	0	0	3	2	2	2	3	4	4	4	0	2	3	3
Miscellaneous (fourteen)	1	10	11	11	2	12	14	15	4	11	17	19	2	8	9	10
Computer Programming:																
Total	4	38	40	41	0	6	7	7	0	0	0	0	0	1	1	1
In Departments of:																
Data Processing		22	24	24		1	1	1			0	0		0	0	0
Computer Programming		5	5	5		0	0	0		0	0	0		0	0	0
Miscellaneous (eight)		11	11	12		5	6	6		0	0	0		1	1	1
Computer Technology:																
Total	8	21	22	22	1	9	9	9	0	3	4	3	0	2	2	2
In Departments of:																
Computer Technology		12	13	13	1	2	2	2		0	0	0		0	0	0
Miscellaneous (eight)[1]		9	9	9		7	7	7		3	4	3		2	2	2
Information Systems:																
Total	0	5	6	7	3	7	9	10	2	5	6	6	1	3	4	4
In Departments of:																
Miscellaneous (nine)		5	6	7		7	9	10	2	5	6	6		3	4	4
Information Science:																
Total	0	0	0	0	2	3	6	5	9	7	10	10	8	6	6	6
In Departments of:																
Miscellaneous (six)		0	0	0		3	6	5	9	7	10	10		6	6	6
Systems Engineering:																
Total	0	0	0	0	1	7	7	8	1	12	12	12	1	7	8	7
In Departments of:																
Systems Engineering		0	0	0	1	3	3	3	1	3	3	3	1	0	0	0
Electrical Engineering		0	0	0		2	2	3		3	3	3		3	3	3
Engineering		0	0	0		1	1	1		5	5	5		3	4	3
Industrial Engineering		0	0	0		1	1	1		1	1	1		1	1	1

TABLE I—Reported Degree Programs in Computer Science, Data Processing, Information Science, etc. for Academic Years 1966-67, 1969-70, 1970-71, and 1971-72—Continued

Miscellaneous (seven):																
Total		1	1	1		5	6	8	1	5	6	6		3	5	5
In Departments of:																
Miscellaneous (ten)		1	1	1		5	6	8	1	5	6	6		3	5	5
TOTAL	142	302	323	332	72	148	182	206	78	116	137	142	46	73	82	85
Estimated Population Totals[2]	178	405	433	445	90	198	244	276	98	155	184	190	58	98	110	114

[1] ( ) Refer to 69-70, 70-71, 71-72

[2] Reported Totals x (Number Institutions in Population)/(Number Institutions Responding)

committees was supported by National Science Foundation.

Since 1966-67 the numbers of degree programs have doubled with the highest increase nearly threefold at the bachelor's level. Three out of five of the latter programs are called Computer Science and the next largest group (26) are designated as Data Processing. Twenty (20) new programs in Computer Science at the bachelor's level were expected to be started during 1971-72 and the next highest increase was expected to be at the master's level in Computer Science with sixteen new programs.

After the report[8] of the ACM Curriculum Committee for Computer Education for Management has had a chance to be circulated, I expect to see a surge in the master's level programs in Information Systems (Analysis and Design).

The numbers of majors enrolled in undergraduate programs have nearly tripled and the numbers of graduates per year have increased fourfold between 1966-67 and 1969-70 (see Table II). Figure 1 shows the growth trends at present. I do not expect significant changes in these trends for the next three to five years. After 1975 I believe that we can expect to see some

TABLE II—Reported Majors Enrolled for 1966-67 and 1969-70 and Degrees Awarded for 1966-67, 1969-70 and 1970-71

	Undergraduate								Graduate							
	No. Majors		Associate			Bachelor's			No. Majors		Master's			Doctorates		
Program Name	66-67	69-70	66-67	69-70	70-71	66-67	69-70	70-71	66-67	69-70	66-67	69-70	70-71	66-67	69-70	70-71
Data Processing	12815	27712	872	3285	3971	74	286	395	317	159	63	9	19	1	0	2
Computer Science	3187	10909	62	259	435	355	976	1537	2926	4725	446	892	1249	133	187	198
Computer Programming	281	4786	78	688	819	0	12	32	0	63	0	0	0	0	0	0
Computer Technology	496	2791	13	218	295	20	60	88	0	53	0	4	9	0	2	2
Information Systems	737	1198	0	66	94	19	91	133	31	241	6	24	43	0	1	6
Information Science	100	546	0	0	0	0	28	78	627	240	64	34	66	3	11	13
Systems Engineering	113	893	0	0	0	7	143	149	48	402	15	85	98	2	10	14
Miscellaneous (seven)	—	356	—	0	0	—	52	107	—	925	—	28	60	—	18	20
Total	17729	49191	1025	4516	5614	475	1648	2519	3949	6808	594	1076	1544	139	229	255
Estimated Population Totals[1]	22161	65916	1281	6051	7523	594	2208	3375	4936	9123	742	1442	2069	174	307	342

[1] Reported Totals x (Number Institiutions in Population)/(Number Institutions Responding)

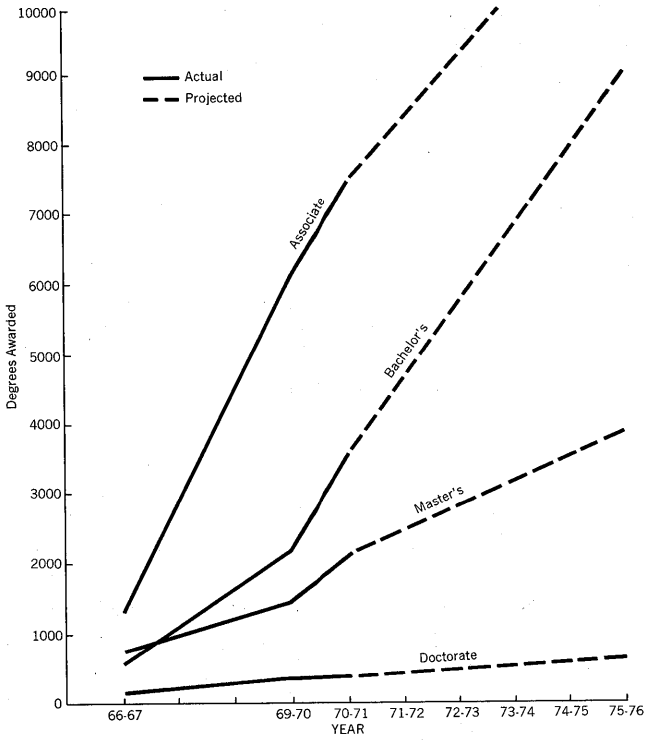


Figure 1—Degrees awarded in computer sciences, data processing, information sciences, etc. (Estimates)

leveling off at all levels. Table III shows my estimates of degrees to be awarded during the year 1974-75.

When these figures of supply are compared with estimates[4] of demand we see that there is no longer a need to encourage a crash effort to start new degree programs at any level. However, if we examine the course offerings of the associate and bachelor's degree programs, in particular, as I have had occasion to do in the two NSF Inventories,[7] there is definitely a need to strengthen these programs both in facilities available and course offerings. During 1966-67 about one out of every two programs at the associate and bachelor's degree levels were judged (by me) to be lacking in either course offerings or computer facilities.[7] Preliminary investigation of data reported for 1969-70 indicate that percentagewise there has been little, if any, improvement when advances in technology are considered.

TABLE III—Estimates of Degrees to be awarded in Computer Sciences, Data Processing, Information Science, Information Systems, etc. During 1974-75

Level	No.	To Enter Manpower Pool	Continue Education
Associate	11,000	9,000	2,000
Bachelor's	8,000	5,000	3,000
Master's	3,500	2,500	1,000
Doctorate	500	500	—
Total	23,000	17,000	6,000

TABLE IV—Estimated Manpower Utilization by Academic Departments for Programs in Computer Science, Data Processing, Information Science, etc. 1966-67

Highest Educational Level:	No.	Percent
High School Diploma	200	7
Associate Degree (2 yr.)	80	3
Bachelor's	770	27
Master's (Incl. 1st Prof.)	960	33
Doctorate	870	30
Total	2,880	100
Job Category:	No.	Percent
Faculty	2,180	76
Prof (non-Fac)	380	13
Other	320	11
Total	2,880	100

MANPOWER UTILIZATION

Computer manpower utilization in institutions of higher education fall under three broad categories. One

TABLE V—Reported Manpower Utilization by Academic Departments for Programs in Computer Science, Data Processing, Information Science, etc. 1969-70

Staff Type	Number People	Number FTE
Full-time	2,150	2,150
Part-time:		
Research Assts.	729	
Teaching Assts.	1,129	
Other	1,578	1,292
Total	5,586	3,442

	Position		
Doctorates	Faculty	Other	Total
Computer Science	345	41	386
Other	785	102	887
Total	1,130	143	1,273

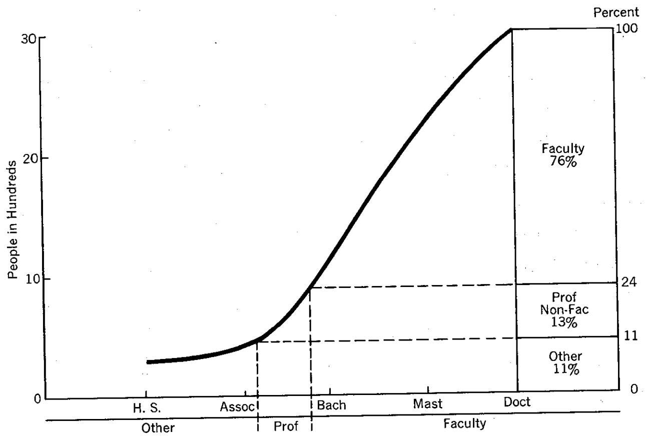


Figure 2—Estimated manpower utilization by academic departments for programs in computer sciences, data processing, information sciences, etc. 1966–67 highest level of education attained and job category

is the personnel of academic departments offering degree programs. For this category we have good estimates of the numbers of persons employed for 1966-67 and 1969-70 (see Tables IV and V and Figure 2). It is of interest to note that only about one out of every two faculty persons hold a doctorate and of those who do

TABLE VI—Estimated Manpower Utilization by Computer Centers in U. S. Colleges and Universities 1966-67

Highest Education Level	Estimated Total	Percent of Total
Other	210	1
High School Students	160	1
High School Diploma	4,890	34
Associate Degree	640	4
Undergraduate	3,920	27
Bachelor's	2,860	20
Master's (Incl. 1st Prof.)	1,260	9
Doctorate	570	4
Total	14,510	100
	Estimated Total	Percent of Total
Job Classification	FTE	
Management	1,970	14
Analysts	1,040	7
Systems Programmer	1,350	9
Applications Programmer	2,360	16
Operators	5,570	39
Clerical	2,220	15
Total	14,510	100

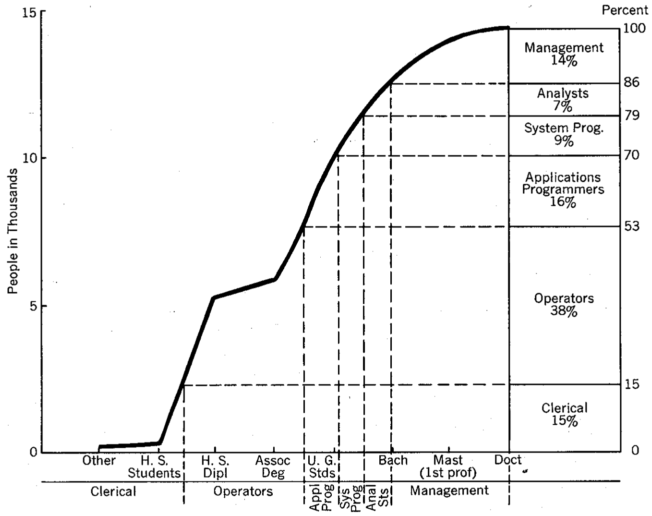


Figure 3—Estimated manpower utilization by computer centers in U. S. colleges and universities 1966–67 highest level of education attained and job classification

TABLE VII—Estimated Manpower Utilization by Computer Centers in U. S. Colleges and Universities 1969-70

Highest Education Level:	Estimated Total	Percent of Total
High School Diploma	9,496	36
Associate Degree	1,360	5
Undergraduate Students	6,928	26
Bachelor's Degree:		
Computer Science	389	1
Other	5,046	19
Master's Degree:		
Computer Science	426	2
Other	2,054	8
Doctorate:		
Computer Science	174	1
Other	829	3
Total	26,704	100
	Estimated Total	Percent of
Job Classification:	FTE	Total
Management	3,111	14
Analysts	1,820	8
Systems Programmers	2,131	9
Application Programmers	4,186	18
Operators	8,135	35
Clerical	3,725	16
Total	23,108	100

TABLE VIII—Estimates of Computer Manpower Utilized by Institutions of Higher Education in other than Computer Centers or Academic Departments Offering Degree Program in Computer Science, Data Processing, etc.
1969-70

Job Classification:	No. of People
Applications Programmers	2,000
Other	1,000
TOTAL	3,000

only about one of four have a doctorate in computer science, information science, etc. This is not surprising because the supply of doctorates in this area has been too small to fill the need. However, as the supply increases, replacements will be made from their ranks rather than from other fields.

The second category of computer manpower utilization is that of personnel required to man the computer facilities of the institutions. This includes facilities for all types of uses—research, administration and instruction. Since similar data was collected for both 1966-67 and 1969-70 some comparisons can be made or trends noticed. Table VI and Figure 3 present these data for 1966-67.

Table VII shows estimated manpower utilization for 1969-70 in the computer facilities of institutions of higher education.

The third category of computer manpower is very difficult to estimate. These are mostly applications programmers, some operators and maintenance personnel who are scattered among the various departments (both academic and administrative) and are not on a computer center payroll. My estimates of such personnel are shown in Table VIII.

SUMMARY

The numbers of degree programs, majors and degrees awarded will be of some surprise to many who are proposing to begin new programs. Of course good, solid offerings will always be desirable. However, in terms of national emphasis it is apparent that we do not need to have a national effort to create new programs. On the other hand, there is a definite need to encourage and assist with the improvement of existing programs at all levels. As a starting point the AFIPS Education Committee sponsored a meeting at the 1971 Fall Joint Computer Conference on "Accreditation: What it is, Who does it, and How it is done," for leaders of the various curriculum committees, special interest groups, and others involved with curriculum planning for Computer Science, Data Processing, Information Science, etc. This was done with the belief that we can work with the various accreditation associations to help bring about the improvements needed in these programs. Government agencies such as the Office of Computing Activities of the National Science Foundation can offer invaluable assistance by providing some of the resources required for the improvement of these programs.

REFERENCES

1 J W HAMBLEN
Computers in higher education: Expenditures, sources of funds and utilization for research and instruction 1964-65 with projections for 1968-69
Southern Regional Education Board Atlanta Georgia 30313 1967

2 J W HAMBLEN
Inventory of computers in U. S. higher education 1966-67: Utilization and related degree programs
Superintendent of Documents U S Government Printing Office Washington D C 1970 NS1 2 C75

3 J W HAMBLEN
Inventory of Computers in U. S. higher education 1969-70
Superintendent of Documents U S Government Printing Office Washington D C 1972

4 B GILCHRIST R E WEBER
Employment of trained computer personnel—A quantitative survey
Proceedings of 1972 SJCC American Federation of Information Processing Societies Inc Montvale N J

5 B GILCHRIST R E WEBER
Sources of trained computer personnel—A quantitative survey
Proceedings of 1972 SJCC American Federation of Information Processing Societies Inc Montvale N J

6 *Curriculum 68: Recommendations for academic programs in computer science*
A Report of the ACM Curriculum Committee on Computer Science Communications of the ACM New York New York Vol 11 No 3 March 1968

7 J W HAMBLEN
Using computers in higher education: Past recommendations, status, and needs
Communications of the ACM New York New York Vol 14 No 11 pp 709-712 November 1971

8 *Professional education in information systems development*
A Report of the Curriculum Committee on Computer Education for Management To appear in Communications of the ACM New York New York Spring 1972

Sources of trained computer personnel—A quantitative survey*

by BRUCE GILCHRIST and RICHARD E. WEBER

American Federation of Information Processing Societies, Inc.
Montvale, New Jersey

INTRODUCTION

The economic recession of the past two years has emphasized the need for better overall manpower planning in the computer field. However, such planning can only be accomplished if there are reasonably reliable data on both the sources of trained personnel and the industry needs for such personnel. The purpose of this paper is to summarize our current state of knowledge of the primary sources of personnel. A companion paper in these proceedings discusses the requirements aspects.[1] It is clear from these two papers that our knowledge of both aspects is limited as to accuracy as well as coverage. For the present this lack of knowledge makes it impossible to construct a manpower model suitable for prognostication. However, the growing interest of the various groups involved in data collection gives rise to optimism that a usable manpower model can be constructed in the foreseeable future.

No attempt is made in this paper to distinguish between the people who will be employed by computer users and those who will be employed by the computer, peripherals and software suppliers. This is due to the lack of data giving specific employments. There is good evidence, however, that the vast majority will be employed by the users of computers. It should also be emphasized that only the primary sources of trained personnel are considered. Promotions from within and movement between companies are important components of any manpower model but here again the lack of data prevents the inclusion of this level of detail at the present time.

In the following sections we review on a type by type basis the available quantitative data on the various sources of trained computer personnel. It should be kept in mind throughout that there are difficulties with regard to definitions. In the absence of generally accepted industry standards, there are many different uses of the titles "programmer" and "systems" analyst and many different assumptions as to what knowledge is required by holders of such titles. The same is true of expressions such as "computer science" and "data processing" as well as of the training or education needed to be eligible to enter one of the computing jobs as a trainee. These definitional problems make comparisons between various reports difficult, if not impossible. We believe, however, that by summarizing in one place all available data, we will point up the estimation, definitional and other problems and thereby encourage improved collection and standardization. This should in turn result in future compilations of data being more internally compatible. Meanwhile we believe that these figures give an idea of the order of magnitude of the output of the various training institutions.

SOURCES OF TRAINING

Public secondary schools

The only comprehensive study at this level is one by the American Institutes for Research under a grant from the National Science Foundation.[2] This study for the school year 1969-70 included a combination mail and interview survey of 23,030 public secondary schools; 12,396 (53.8 percent) of the schools responded in some form.

While the report contains a wealth of information, the question "how many students in each school were receiving instruction in the use of computers" was not specifically asked. Some interesting conclusions can, however, be drawn from the report.

Excluding guidance applications, 464 (or 12.3 per-

* This paper was prepared as part of the AFIPS Statistical Research Program which is partially funded by the National Science Foundation under Study Grant GJ 996.

TABLE I—Projections of Students Receiving Computer Training in Public Secondary Schools during 1970-71

Name of Course	Percent of Sample Population* Enrolled in Course	AFIPS Estimate of Total USA Students Enrolled in Course	
Some Computer Training			88,500
Introduction to Data Processing	.38	66,000	
Computer Operation	.01	1,700	
Computer Processing Equipment Operation	.02	3,400	
Computer Mathematics	.12	17,400	
Computer Programming	.07		12,100
Keypunch Operation	.02		3,400
Total	.62		104,100

* Since this is a count of course enrollments, it does not represent individual pupils who may be enrolled in two or more courses simultaneously.
Data based on Office of Education pretest sample

cent) of the 3,770 schools responding to the main questionnaire reported that they gave instruction in EDP skills. (If guidance and administrative users are included this usage increases to 34.4 percent of all respondents.) While there were wide variations among schools in the number of students involved in each instructional application, the median was 50 students, each of whom participated on the average about ten hours per month. Assuming that the mean number per school is equal to this median, extrapolation to the total of 23,030 schools gives

$$\frac{464}{3770}\times 23{,}030\times 50=142{,}000$$

students receiving *some* instruction in EDP skills during the school year 1969-70.

Furthermore, the report indicates that 3 percent of schools not then offering computer instruction were planning to do so and 22 percent of those giving instruction expected to expand their instructional activities. It is probable, therefore, that enrollment for training in EDP skills in 1970-71 could have reached or exceeded 200,000 students. Based on interviews at some schools, one of the authors of the report estimates that 10 percent or 20,000 of these get sufficient instruction to be hired directly upon graduation as programmer trainees. From the employment viewpoint, we are mostly concerned with this 20,000. Of course, many of them will not look for such jobs but instead go on with their education or go into other fields.

The numbers given above all relate to student enrollments. To obtain an estimate of how many 1971 high school *graduates* had received computer instruction we use the New Jersey statistic that 21.5 percent of students in grades 9-12 graduate each year.[3] This results in an estimate that, of the students graduating from high school in June 1971, 43,000 had some computer training and of these 4,300 had significant programmer training.

Additional information on the number of the graduates will be forthcoming from the U.S. Office of Education. They are planning a sample mail survey of public schools having at least one of grades 7-12 in order to gather course and curriculum information as well as enrollments for the 1972-73 school year. In preparation for the main survey, a pretest of the questionnaire to be used was sent to a sample of 290 out of the 25,128 public schools on their 1968-69 list. The schools were arranged by size and then randomly sampled *within* each state with a minimum of 3 schools per state as the beginning point for each state.

Table I is a summary of some results from this test.[4]

The Office of Education figures are about half those inferred from the American Institutes for Research report. Each survey has its advantages and drawbacks so, for lack of further data, we average the two results to obtain an estimated 1971 graduating class of 33,000 with *some* computer training of which 3,300 have *significant* programmer training.

In order to arrive at job entry estimates we use New Jersey data to obtain percentage distributions of what happens to high school students after they graduate. Applied to the above figures, this results in Table II which indicates that about 7,600 entered the labor force in June 1971. The big assumption made here is that students with *some* computer training will follow the same post high school pattern as the average student.

TABLE II—Estimate of Destination of Students with *Some* Computer Training Graduating from Public Secondary Schools in June 1971

Destination	N.J. percent distribution	Number
Technical School	8	2,600
Junior College	15	5,000
College	41	13,500
Other	13	4,300
Employment	23	7,600
Totals	100	33,000

TABLE III—Students Enrolled in Computer Occupational Courses in Public Vocational Schools for the Years 1966-70

Type of Training	1966	1967	1968	1969	1970
Business Data Processing	42,764	85,063	109,769	134,723	165,977
Scientific Data Processing	—	26,367	26,374	35,914	18,162*
Total	42,764	111,430	136,143	170,637	184,159*

* Beginning in 1970, States with less than 5 percent of their enrollments in a particular category were permitted to include this number in a general category. This threw many of enrollments previously reported under scientific data processing into the general category. The 1970 figure is therefore not comparable with previous years nor necessarily indicative of the true picture.

It should be emphasized that the foregoing estimate considers only public secondary schools. In addition to these, there are approximately 4,000 private and parochial secondary schools with enrollments of about 1.4 million students.[5] If the same proportion of these students as of students in public secondary schools receive computer instruction, their inclusion would increase the numbers in Table II by about 8 percent.

Public vocational schools

Under the ongoing vocational education program the U.S. Office of Education requires each state to make annual reports of their enrollments in vocational programs. This data is then summarized and published by the Office of Education. Table III gives their latest data.[6]

Table III, excluding the incompatible 1970 figure, shows a fairly consistent annual growth rate. Linear extrapolation is therefore used to project the enrollments for 1971. Using these estimates and information from follow-up records provided by the U.S. Office of Education, Table IV has been prepared to show estimates of the numbers graduating from public vocational programs in June 1971 and of those entering the labor force and higher education.

Although it is impossible to give the exact numbers entering each occupation, it has been observed that the Business Data Processing program is intended to prepare students for entry into employment as computer programmers, computer operators or keypunch operators; whereas the Scientific Data Processing program prepares students for jobs as systems analysts and computer programmers.

It should be noted that these vocational programs are offered in comprehensive high schools, community colleges and four-year colleges, as well as specialized vocational schools. Thus, it is possible that there is some duplicate counting between this section and other parts of this report. It is believed that this duplication is minimal.

TABLE IV—Estimate of Students Enrolled in and Completing Vocational Programs and Number Placed in 1971

				Destination		
					Entering Labor Force	
Training and Level	Total Enrollment	In Preparatory Programs	Total Graduates	Entering Higher Education	Total	Field For Which Trained
Business Data Processing	196,800	131,200	36,736	5,640	21,674	17,439
Secondary		52,480	13,120	3,280	6,560	4,592
Post Secondary		78,720	23,616	2,360	15,114	12,847
Scientific Data Processing	45,500	30,334	8,691	1,150	5,301	4,336
Secondary		7,583	1,866	450	933	653
Post Secondary		22,751	6,825	680	4,368	3,713
Total	242,300	161,534	45,427	6,790*	26,975*	21,805

* In addition to these two groups there were 11,662 that went into the armed services, married, became ill, etc., and were thus unavailable

TABLE V—Summary of Responses from 41 Private EDP Schools

Type of Course	Number of Courses	Median Course Length (Hrs.)	Expected 1969 Graduates
Programming	44	500	7,994
Computer Operations	17	180	1,446
Keypunching	14	78	4,359
Other	5	—	1,200
Totals	80	—	14,999

Private vocational schools

Due to the general shortage of programmers in the 1960's, a large number of so-called private EDP schools were begun. These schools advertised heavily and appealed to non-college graduates. Such schools are part of a larger set of private trade and technical schools.

Some of these schools are accredited by organizations such as the National Association of Trade and Technical Schools (NATTS) and in some states there are regulations governing various aspects of their operations. However, there are no nation-wide standards with respect to course length or content and no reporting scheme for enrollment and graduation data.

In 1969 Gilchrist made a mail survey of 207 private EDP schools located in 10 major U.S. cities.[7] The questionnaire asked for the number, type and length of courses offered together with the estimated number of graduates from such courses in 1969. The 41 schools which responded reported that they expected to graduate nearly 15,000 students that year as shown in Table V.

Using the sample data, Gilchrist then extrapolated linearly to the total population of private EDP schools which he estimated to be between 500 and 1,000. This resulted in the figures summarized in Table VI.

TABLE VI—Estimated Numbers of Private EDP School Graduates for 1969

Type of Course	Sample (41)	Low Estimate (500)	High Estimate (1000)
Programming	7,994	97,500	195,000
Computer Operations	1,446	17,600	35,000
Keypunching	4,359	53,200	106,000
Other	1,200	14,600	29,000
Totals	14,999	182,900	365,000

Belitsky, using 1966 data, found that there were then 3,000 private trade and technical schools with an average enrollment of 280 students and that data processing courses represented 22 percent of all vocational courses.[8] He also found that there were over 500 correspondence schools for vocational training. These results indicate that the correct number of graduates probably does lie between Gilchrist's high and low estimates.

Since 1969 private EDP schools appear to have contracted both in number of schools and in number of students per school. This is due to the slowing of the economy, the difficulties their graduates, especially in programming, are encountering finding jobs and the public disclosure of poor practices in some of these schools. While no survey results are available, limited data from New Jersey indicate that there could have been a 50 percent contraction since 1969. In New Jersey the number of private data processing schools for the years 1968 through 1971 were 52, 52, 40 and 28, respectively.[3] Allowing for some contraction Table VII uses 350 schools each having 225 graduates to estimate the 1971 output of the private EDP schools.

TABLE VII—Estimated Numbers of Private EDP School Graduates for 1971

Type of Course	Estimated Number of 1971 Graduates
Programming	43,000
Computer Operations	7,400
Keypunching	22,000
Others	6,600
Total	79,000

The nature of the courses given in private EDP schools and the whole flavor of their promotional advertising indicates that *all* their graduates enter the labor market rather than going on to higher education.

Higher education

In the area of higher education the U.S. Office of Education publishes annual reports on the number of graduates by discipline and by degree level.[9–10] These reports include a number of classifications relating to the computer field. In addition, the National Science Foundation has sponsored an information gathering project at the Southern Regional Education Board. This project has included surveys of the uses of computers in higher education during the school years 1964-65, 1966-67 and 1969-70. Full reports are available

TABLE VIII—Comparison between U.S. Office of Education and Southern Regional Education Board Classifications for Computer Related College Programs

Office of Education Classification	SREB Classification
Systems Analysis	Systems Analysis Information Systems Information Processing Systems Engineering
Computer Science	Computer Science Information Science Management Science Option in Electrical Engineering Option in Mathematics Option in Engineering Option in Industrial Engineering
Data Processing	Data Processing Computer Technology Computer Programming Option in Business Administration
Other	Classifications not listed above

for the first two surveys[11,12] and a preliminary report on the most recent one is included in these proceedings.[13]

Unfortunately, the Office of Education reports and the Southern Regional Education Board reports of 1966-67 and 1969-70 use somewhat different classifications. In order to compare data from the three sources we have grouped the various SREB classifications into four major groups which we believe to be sufficiently comparable to the Office of Education classifications. This grouping is shown in Table VIII.

Table IX was prepared by grouping the SREB results according to Table VIII and extrapolating to the total population of institutions for each year surveyed.

The very low ratio of graduates to majors for 1966-67 shown in Table IX indicates that in that year many programs were still in their early stages of operation. By 1970-71 this ratio had become more normal indicating that most programs were then in full operation. Even so, without an increase in enrollments, the number of graduates can still be expected to increase somewhat for the early 1970's. The increasing number of degrees awarded in computer related courses is confirmed by the U.S. Office of Education reports which are summarized in Table X.

The U.S. Office of Education also publishes data on the number of students enrolled for advanced degrees.[14] For the computer related disciplines, these data are summarized in Table XI. The 1967 numbers for "Systems Analysis" and "Other" categories look strange and suggest that definitional problems resulted in "Systems Analysis" students being put into the "Other" category.

The growth rates are remarkably consistent. They also indicate that the level of graduations at the bachelors and masters levels appear sufficient to continue the respective master and doctoral enrollments growing at substantial rates.

As a result of the aforementioned classification differences, comparisons from Tables IX, X, and XI leave something to be desired. However, considerable growth is implicit in Table IX and quite obvious in Tables X and XI. Regression analysis of the data from Tables X and XI reveals that the growth rates for nearly all categories were approximately constant. The mean growth rates are given in in Table XII.

The question now arises as to how many of these 1971 graduates entered the labor force in contrast to continuing their education. There appear to be no data upon which to base an answer to this question. A very rough approximation might be that a third of those receiving lower degrees continued their education

TABLE IX—Summary Results Derived from 1966-67 and 1969-70 Southern Regional Education Board Surveys

	Undergraduates						Graduates					
	Majors		Associate		BA/BS		Majors		MA/MS		PhD	
Major Field	66-67	69-70	66-67	70-71	66-67	70-71	66-67	69-70	66-67	70-71	66-67	70-71
Systems Analysis	1060	2800	—	125	30	340	100	850	25	185	3	25
Computer Science	4150	15350	80	570	450	2170	4445	6650	635	1770	170	280
Data Processing	16950	47200	1200	6800	110	700	395	370	80	40	1	5
Other	—	475				140		1250		75		25
Total	22160	65925	1280	7495	590	3350	4940	9120	740	2070	174	335

TABLE X—Number of Graduates of Computer Related Programs from 1965 to 1970 as Reported by the U.S. Office of Education

Degrees Conferred for Years 1965-70																	
	Field and Level																
Year	Data Processing				Computer Science			Systems Analysis			Other			Total			
	Associates		BA/ BS	MA/ MS	BA/ BS	MA/ MS	PhD	BA/ BS	MA/ MS	PhD	BA/ BS	MA/ MS	PhD	Assoc	BA/ BS	MA/ MS	PhD
	Scien- tific	Busi- ness															
1964-65	—	—	25	5	6	57	5	—	34	—	36	50	1	—	67	112	6
-66	—	—	37	19	22	157	14	30	26	—	—	36	5	556	89	248	19
-67	—	—	48	24	71	239	20	48	101	12	15	85	6	1009	232	449	38
-68	503	2405	155	47	166	317	20	92	50	5	46	134	11	2908	459	548	36
-69	1222	3351	239	73	463	622	44	179	103	5	52	214	15	4633	933	1012	64
-70	1627	4860	336	43	825	689	79	195	226	2	188	501	25	6487	1544	1459	106

and that two thirds of those receiving PhD's have gone into teaching. Applying these approximations to Table IX we get the estimate of the numbers entering the labor force in June 1971. These are given in Table XIII.

Summary of formal education programs

The data from the previous four sections are summarized in Table XIV.

With the exception of the Private Vocational Schools each program is producing a steadily increasing number of graduates and entrants to the labor force. There is, unfortunately, insufficient data to break down the 1971 entrants to the labor force into job skills. However, one can roughly say that the first three categories in Table XIV will primarily enter data-entry, electrical accounting machine and computer operations while the last three categories will primarily enter programming and systems analysis. The holders of Associate degrees will divide—the top students going into programming and the others into machine operations.

Other sources

In addition to the estimated 122,000 entrants to the labor force from the formal computer related programs discussed above, a significant number of people are trained by other sources. These include manufacturers of computing equipment and software as well as large

TABLE XI—Number of Students Enrolled for Advanced Degree in Computer Related Programs, 1965–1970

Year	Data Processing	Computer Science	Systems Analysis	Other	Total
Fall 1965	37	323	185	271	816
Fall 1966	55	587	517	873	2,032
Fall 1967	51	962	84	1,625	2,722
Fall 1968	159	1,888	627	1,217	3,893
Fall 1969	290	3,906	601	1,404	6,201
Fall 1970	224	4,804	1,213	1,695	7,936

TABLE XII—Mean Annual Percentage Growth Rates for Number cf Degrees Awarded in Computer Related Programs and for Enrollments for Advanced Degrees in Such Programs

Level	Data Processing	Computer Science	Systems Analysis	Total
Associate	64	—	—	64
BA/BS	56	99	45	67
MA/MS	44	48	37	49
PhD	—	49	—	51
Enrolled for Advanced Degree	52	62	—	47

TABLE XIII—Approximate Numbers of College Graduates Entering the Labor Force, Excluding Academic Personnel in June 1971

Degree	Number Entering Labor Force
Associate	5,000
BA/BS	2,300
MA/MS	1,400
PhD	110

users. For example, in 1970 the U.S. Defense Department trained about 2,000 of their own personnel. Also, several insurance companies are known to have obtained state licenses to give courses for computer user personnel.

Little information exists with regard to the output of such sources of trained pesonnel. The importance of these training programs is illustrated by Table XV which is derived from the *1971 AFIPS Personnel Survey* which sampled the members of the AFIPS constituent societies.[15]

Most of the people trained by these other sources have probably had some prior training, perhaps in the secondary or vocational schools or Associate degree programs. Since we are here concerned mainly with primary sources, it is unclear just how to treat these other training sources. Our investigations suggest that in past years between 20,000 and 30,000 people were trained in this way, but that the figure for 1971 will be from 10,000 to 20,000. It is to be expected that as the level of training offered by the public schools and two year colleges is increased, training by manufacturers and users will become numerically less important insofar as primary sources are concerned.

The computer manufacturers' courses and in-house use training courses are also very important from the viewpoint of continuing education. In a rapidly developing field such as computing there will continue to be a great need for short refresher courses and courses on new developments. Some of these will be given by colleges, universities and companies devoted to training, but we suspect that the majority will continue to be provided by manufacturers and users.

TABLE XIV—Estimates of Number of Entrants into the Labor Force from Various Formal Computer Educational Programs in 1971

Educational Program	1971 Graduates	Entering Labor Force
High School	33,000	7,600
Public Vocational School	45,400	27,000
Private Vocational School	79,000	79,000
Associate	7,495	5,000
BA/BS	3,350	2,300
MA/MS	2,070	1,400
PhD	335	110
Totals	170,650	122,410

TABLE XV—Sources of Information Processing Training Reported by Respondents to the 1971 AFIPS Personnel Survey

Source	Percentage*
Computer Manufacturers	47.1
In-House	44.5
Part of Formal Education	37.5
University/College	31.4
Software Consulting Co.	8.5
EDP School	6.0
Correspondence School	4.5
High School/Vocational School	1.5
Other	9.2

* Percentages add up to more than 100 due to multiple responses.

SUMMARY

The available data, supplemented by personal interviews, indicate that at least 170,000 people were trained by primary sources for the computer user labor market in 1971. The level of training is clearly heavily weighted toward the low end. It also appears that there is a trend toward higher level education together with a move toward general rather than specific training. It is further noted that there is a decided shift toward education in publicly supported institutions as opposed to private vocational and manufacturer's schools.

The estimated number of trained people entering the labor force does not appear proportionate to the requirements suggested by an employment figure for their occupational categories of the order of slightly over a half million. With the reduced growth rate being experienced by the computer industry an implied new hire ratio of nearly 30 percent seems entirely unlikely. Furthermore, preliminary indications are that the numbers of people trained for various levels are not commensurate with the needs of the industry at the respective levels.

In view of these conclusions, it is clear that a strong need exists for improved data collection followed by public release of accurate findings as to potential training outputs and job requirements at the various skill and educational levels. To this end measures should be taken by all concerned parties to initiate such projects. Thereby we may eliminate future imbalances and thus improve labor conditions in the computer field.

ACKNOWLEDGMENTS

The authors are very appreciative of the many individuals, both in industry and government who have helped them track down sources of data and who commented upon a working paper which formed the basis of this report.

REFERENCES

1 B GILCHRIST R E WEBER
Employment of computer personnel—A quantitative survey
Proceedings of the Spring Joint Computer Conference
Vol 40 1972

2 C A DARBY JR A L KOROTKIN T ROMASHKO
Survey of computing activities in secondary schools, final report
American Institutes for Research Pittsburgh Pa Oct 1970

3 Various reports and interviews New Jersey State
Education Department Trenton NJ

4 *Pretest survey of public and non-public secondary school offerings, enrollments, and curriculum practices, 1970-71*
(Unpublished) US Office of Education Washington DC

5 *Statistical abstract of the United States*
US Department of Commerce Washington DC 1970

6 *Enrollment in vocational education occupational programs*
US Office of Education Washington DC 1971

7 B GILCHRIST
A pilot study of the contribution of private data processing schools to the manpower pool of the information processing industry
AFIPS Montvale NJ 1969

8 A H BELITSKY
Private vocational schools: Their emerging role in post secondary education
Staff Paper Upjohn Institute Wash DC 1970

9 *Earned degrees conferred*
Annual Reports US Office of Education Washington DC
1965-70

10 *Associate degrees and other formal awards below the baccalaureate*
Annual Reports US Office of Education Washington DC
1965-70

11 J W HAMBLEN
Computers in higher education, expenditures, sources of funds and utilization for reserved and instruction 1964-65 with projections for 1968-69
Southern Regional Education Board Atlanta Ga Aug 1967

12 J W HAMBLEN
Inventory of computers in U.S. higher education 1966-67
Southern Regional Education Board Atlanta Ga 1970

13 J W HAMBLEN
Production and utilization of computer manpower in U.S. higher education
Proceedings of the Spring Joint Computer Conference
Vol 40 1972

14 *Students enrolled for advanced degrees*
Annual Reports US Office of Education Washington DC
1965-70

15 *Summary report of the 1971 AFIPS information processing personnel survey*
AFIPS Press Montvale NJ Oct 1971

Employment of trained computer personnel—A quantitative survey*

by BRUCE GILCHRIST and RICHARD E. WEBER

American Federation of Information Processing Societies, Inc.
Montvale, New Jersey

INTRODUCTION

After several years of chronic shortages of trained personnel, the economic recession of the past two years has resulted in some unemployment within the computer field. This unemployment has emphasized the long existing need for better overall manpower planning. Such planning can only be carried on successfully if reasonably reliable data and projections are available. The information collected and the resulting projections must include both the employment needs and the sources of trained personnel. The purpose of this paper is to summarize the current employment picture especially as it relates to the users of computers. A companion paper in these proceedings discusses the primary sources of such trained personnel.[1]

In the past, a number of general statements have been made about the employment of computer personnel. For example, the Bureau of Labor Statistics regularly publishes occupational outlook data for several computer related occupations,[2] and industry leaders are prone to making generalized comments. However, as was pointed out by Gilchrist in 1969, these numbers appear, upon inspection, to be based on relatively little firm data.[3]

AVAILABLE DATA SOURCES

The 1970 decennial census included, for a 20 percent sample of the population, a question on occupation. The encoders of these responses had available to them a number of classifications relating to computing. The summarized results should give an interesting picture of employment in the computer field. Unfortunately, the summaries will not be current when published since they will not be available until late 1972 at the earliest.

In the meantime, a number of groups have ongoing activities or are commencing new ones which should improve our knowledge of the employment picture.

AFIPS with partial support from the National Science Foundation is assembling available data and encouraging trade associations to obtain additional data from their individual member firms.

The Federal Government has for a number of years reported their EDP related employment figures. Now various states and cities are beginning to produce similar reports on their own or in conjunction with the Bureau of Labor Statistics (BLS) of the U.S. Department of Labor.

The BLS has recently included several computer related occupations in its regular *Area Wage Surveys*.[4] Although these data are collected primarily to determine wage patterns, they do provide estimates of employment in each occupation. With financial support from the National Science Foundation, BLS has recently commenced a major study of computer related employment. In addition, it has made a number of relevant studies of employment in individual cities and industries.

On a monthly basis, BLS publishes employment and earnings data covering every industry in the country. The difficulty in using this data to learn about computer related occupations is that the SIC classifications reflect the products of a company and not the type of employees. For example, programmers employed by a chemical company will be reported under the SIC code for the particular chemical products. Plants whose prime output is computer equipment are identified separately and employment data on these are available.

* This paper was prepared as part of the AFIPS Statistical Research Program which is partially funded by the National Science Foundation under Study Grant GJ 996.

TABLE I—Man-years Utilized in Federal Government ADP Operations

Fiscal Year	ADP Man-years
1960	48,700
1961	54,400
1962	58,800
1963	67,400
1964	72,300
1965	76,200
1966	89,600
1967	105,900
1968	118,800
1969	118,900
1970	127,500

SUMMARY OF DATA

Despite these varied activities, we can still paint only a hazy and incomplete picture of the employment situation in computer related occupations. In the following sections we summarize, source by source, the currently available data. It must be recognized throughout that there are problems arising from the lack of widely accepted standard definitions for occupations such as programmer, systems analyst, computer operator, etc. This makes exact comparisons between data from different sources very difficult. In addition, there are a growing number of individuals who spend part of their time programming and part of their time performing other tasks such as engineering or accounting. How these people are classified in the various surveys is unknown. This problem will undoubtedly get worse with the increasing use of terminals which make computers available to many more people who would not normally regard themselves as computer programmers or computer operators.

TABLE II—Distribution of Man-years Utilized in Federal Government ADP Operations

Occupation	1967	1968	1969	1970
Systems Analysis and Design	8,027	10,533	12,006	12,921
Programming	15,492	16,752	17,086	18,552
Operations	33,665	35,585	34,591	36,769
Keypunching	23,536	25,062	22,826	23,126
ADP Equipment Selection	696	1,003	843	742
In-House Maintenance	1,043	1,292	1,774	2,152
Services and Support*	23,475	28,573	29,743	33,229
Total	105,934	118,813	118,869	127,491

* In a few agencies a large number of people are engaged in the handling of source documents.

Federal government

As the result of a directive from the Office of Management and Budget, the federal agencies are required to make annual reports on their use of automatic data processing (ADP) equipment.[5] These reports include cost, manpower and utilization data for all computers except those used for control purposes or installed in classified locations. A summary of all the agency reports is published annually. Tables I and II are taken from the report for fiscal year 1970.[6]

Although the numbers are subject to some uncertainty because of a lack of complete conformity in job descriptions the two tables present an interesting and quite detailed picture of Federal Government employment.

State governments

The most comprehensive source of data on employment in state governments is a survey made in 1970 by The Research and Education Committee of the National Association for State Information Systems (NASIS).[7] While the data is by no means complete the majority of the states, including the larger ones, did report computer related personnel figures. These are summarized in Table III which covers employment by 36 states and includes an extrapolation to the complete 50 states. This extrapolation is based on a report that the 36-included states account for 83.8 percent of all full-time equivalent employment by all 50 states.[8]

TABLE III—Employment of Computer Personnel by State Governments for FY 1970

Occupation	36 States*	50 State Extrapolation
Systems Analysis & Design	2,100	2,506
Programming	4,199	5,011
Computer Operations (incl. EAM)	5,832	6,959
Data Entry (Keypunching)	9,565	11,414
Supervision and Administration	1,633	1,949
Total	23,329	27,839

* In a few cases totals were prorated among the various job categories on the basis of overall percentages.

Local governments

A representative picture of computer related employment in city governments can be obtained from studies made between May and October of 1970 by BLS in cooperation with several cities.[9–15] The eight cities covered by these reports account for approximately 9 percent of all local government employees. Table IV summarizes these data and extrapolates them to all local governments.

In addition, New York City has prepared a detailed report on their EDP operations.[16] Table V, which gives the 1969 employment situation in more detail for New York City, is taken from this report and illustrates the importance of their computer operations. (This is far more comprehensive than a recent BLS survey of municipal government workers in New York City.[17])

TABLE IV—Employment of Computer Personnel by Cities for 1970

Occupation	8 Cities	All Local Governments
Systems Analyst	123	1,353
Programmer	276	3,036
Machine Operator (incl. EAM)	359	3,949
Keypunch Operator	846	9,306

Establishments in metropolitan areas

Data from the individual *Area Wage Surveys*[4] for the 90 Standard Metropolitan Statistical Areas (SMSA) have been consolidated and are presented in Table VI. The surveys were conducted at various times from late 1969 through early 1971 and cover about 24,000,000 employees in manufacturing and non-manufacturing. The latter group includes the transportation, communications, public utilities, wholesale and retail trade, finance, insurance, real estate and service industries. Data by industry group is summarized in Table VII. There is a great deal of additional detail in these surveys. For example, employment is broken down by level and by sex.

The twenty-four million employees covered by the surveys are about 42 percent of those on non-agricultural/non-government payrolls.[19] Using this percentage as a basis for extrapolation we obtain Table VIII which gives an estimate of employment in all non-agricultural/non-government establishments.

Definitional problems preclude direct comparison of data in Tables VI and VII with that in the earlier tables. A major problem is that the *Area Wage Surveys* clearly do not include *scientific* and *engineering* programmers and systems analysts whereas these categories are included in the various government figures.

TABLE V—City of New York Employees Performing DP Functions in 1969

Function	Number of Employees
DP Management	31
Systems Analysis	42
Programming Management	19
Programming	176
Operations Management	52
Computer Operations	61
Data Control	160
EAM Operations	147
Keystroke	436
Miscellaneous	68
Total	1,192

Manufacturers of electronic computing equipment

There are two sources of data on total employment by the manufacturers of computing equipment. The first is the Bureau of the Census' *Annual Survey of Manufacturers* which surveys one out of every five known computer firms. Results from this survey are included in Department of Commerce reports on the economy.[18] The second is the Bureau of Labor Statistics' monthly publication *Employment and Earnings*[19] which is based on a sample survey of firms derived from the Social Security Administration's corporate returns. Unfortunately the two sources do not agree, the Census estimates being lower than those of the Department of

TABLE VI—Employment of Computer Personnel in Covered SMSA by Size of Establishment

Occupation	Establishment		
	Large	Small	All
Business Systems Analyst	13,078	20,606	33,684
Business Programmer	14,783	31,179	45,962
Computer Operator	17,451	34,623	52,074
Keypunch Operator	43,750	99,038	142,788
Number of covered employees	6,735,003	17,379,049	24,114,052

TABLE VII—Employment of Computer Personnel in Covered SMSA by Industry

Occupation	Manu-facturing	Non-Manufacturing					
		Total*	T.C.P.**	Wholesale	Retail	R.I.F.**	Service
Business Systems Analyst	13,560	17,824	1,759	328	232	5,626	780
Business Programmer	15,467	28,562	3,254	786	222	9,639	1,539
Computer Operator	18,346	32,977	3,117	1,770	1,138	10,411	1,975
Keypunch Operator	47,870	94,736	13,688	8,752	10,007	17,650	3,874
Number of Covered Employees	11,398,143	12,715,909	—	—	—	—	—

* The sum of the subdivisions do not equal these figures. In many cases the BLS criteria for publication do not permit full subdivision of totals within an SMSA. Clearly, the total figure is the most accurate. Likewise, manufacturing and non-manufacturing do not total to the "all establishment" figure on Table VI and in this case the "all establishment" figure is the more accurate.

** T.C.P.—Transportation, Communications and Public Utilities
R.I.F.—Real Estate, Insurance and Finance

Labor. A further difficulty arises from the fact that data from prior years is frequently corrected by both agencies and these corrections are only reported when new summaries covering past years are published. Moreover, neither of these surveys gives the data broken down by occupational specialty.

Local offices of the Bureau of Labor Statistics will, upon request, give out their latest figures. The figures given in Table IX were obtained from the New York office of the Bureau of Labor Statistics in December 1971.

In late 1970 AFIPS made a survey of the principal manufacturers of CPU's to determine the number of programmers and systems analysts employed by such firms. Data was obtained from five of the manufacturers including the three companies reputed to have the largest share of the CPU market. These companies reported a total of 14,800 programmers and systems analysts as of mid-1970 with academic training as given in Table X.

COMPARISON OF THE DATA

Before trying to make estimates of total employment in the various categories we first look at some averages derived from the foregoing tables. In Table XI we give the average number of employees in each occupation per one thousand employees in the group specified. It should be noted that federal, state and local figures include both *scientific* and *business* systems analysts and programmers while the other groups do not.

Examination of this table reveals both similarities and differences. For example, the total per thousand employment by Federal Government (15.9) is nearly equal to that of the large establishments (13.2). Likewise, the systems analyst and programmer figures for these two groups are nearly equal. Any differences can be rationalized by recalling the omission of scientific personnel from the establishment counts. The large establishment figures need only to be inflated by 33 percent to obtain the federal total for these two occupa-

TABLE VIII—Estimated 1970 Employment of Computer Personnel by Private Non-agricultural/Non-government Establishments

Occupation	Employment
Business Systems Analyst	80,000
Business Programmer	110,000
Computer Operator	125,000
Keypunch Operator	340,000
Total	655,000

TABLE IX—Total Employment by Companies in SIC 3573 (Electronic Computing Equipment) as Reported by BLS

Date	Employment
1967 Average	145,100
1968 Average	160,600
1969 Average	182,700
1970 Average	190,300
Jan 1971	175,800
April 1971	172,800
July 1971	171,500
Sept. 1971	169,500

TABLE X—Educational Background of Programmers and System Analysts Employed by Five Manufacturers of CPU's as of Mid-1970

Level	Percent of programmers and system analysts
PhD	0.5
MA/MS	10.8
BS/BA	58.3
Less than BS/BA	30.4
Total	100.0

tions (i.e., add 1.4 scientific programmers and analysts to the present 4.1 business programmers and analysts). One striking difference to be noted is the high figure for computer operators in the federal as compared to other groups. This may be the result of the job definition problem mentioned earlier. Another difference is that non-manufacturing industries employ more computer personnel on the average than manufacturing industries. We suspect that the finance and insurance segments account for much of this difference. Also to be noted, is that federal, state and local ratios for all occupations are in descending order of magnitude from federal through local with the latter two considerably below the federal levels. This last comparison also holds in absolute numbers as is evident from Tables II, III and IV.

We next examine the ratios of the various occupations to the number of systems analysts. Table XII presents these ratios.

There is a similarity between state and local government in their relative usage of all occupational categories with the possible exception of keypunch operators. One noticeable difference is that small establishments use relatively fewer systems analysts and considerably more keypunch operators than do large establishments. Another is that the Federal Government uses considerably fewer keypunch operators per systems analyst than any other employing group. In this connection it is interesting to note that Table II indicates that the Federal Government is employing a smaller percentage of keypunch operators each year. Table XII further shows that the use of computer personnel by wholesale and retail industries is weighted considerably more toward keypunch operators than other groups; the wholesale industry uses proportionately fewer systems analysts; and the retail group is the only one which employs more systems analysts than programmers.

A final useful comparison can be made between these staffing ratios and similar ratios based on staffing needs per computer. Table XIII presents two such sets of ratios. One is adapted from a British report[20] and a second from information provided in an interview with personnel of a large computer manufacturer.

As can be noted, with the possible exception of computer operators for large CPU's, these ratios are similar to those presented in Table XII. There is also considerable similarity with ratios derived from the 1971 *Business Automation* "EDP Salary Survey."[21] This survey reports on the three categories "Systems Analyst," "Analyst/Programmer" and "Programmer." Ignoring the middle category (or alternatively, assuming that it divides in the same ratio), the ratio of programmers to systems analysts is 1.66. In the 1970 reports this ratio was 1.5.

FINAL ESTIMATE

In order to estimate total U.S. employment in the four occupations, we first accumulate in Table XIV the data for federal, state, local and non-agricultural establishment groups from the preceding sections.

TABLE XI—Average Computer Personnel Per Thousand Covered Employees in SMSA Surveys

	Government*			Establishment			Industry	
Occupation	Federal	State	Local	Large	Small	All	Manu-facturing	Non-Manu-facturing
Business Systems Analyst	2.3	0.9	0.2	1.9	1.2	1.4	1.2	1.4
Business Programmer	3.2	1.8	0.5	2.2	1.8	1.9	1.4	2.3
Computer Operator	6.4	2.5	0.6	2.6	2.0	2.2	1.6	2.6
Keypunch Operator	4.0	4.2	1.5	6.5	5.7	5.9	4.2	7.5
Total	15.9	9.4	2.8	13.2	10.7	11.4	8.4	13.7

* Government figures also include *scientific* and *engineering* systems analysts and programmers

TABLE XII—Ratio of Computer Personnel in Various Occupations to Those in Systems Analysis as Shown in Table XI

Occupation	Government*			Establishment			Industry						
	Federal	State	Local	Large	Small	All	Manufacturing	Non-Manufacturing					
								Total	T.C.P.**	Wholesale	Retail	R.I.F.**	Service
Business Systems Analyst	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Business Programmer	1.4	2.0	2.2	1.1	1.5	1.4	1.1	1.6	1.9	2.4	.96	1.7	2.0
Computer Operator	2.8	2.8	2.9	1.3	1.7	1.6	1.4	1.7	1.8	5.4	3.9	1.9	2.5
Keypunch Operator	1.8	4.6	6.9	3.3	4.8	4.2	3.5	5.3	7.8	26.7	43.1	3.1	5.0

* Government figures also include scientific and engineering systems analysts and programmers
** T.C.P.—Transportation, Communications and Public Utilities
R.I.F.—Real Estate, Insurance and Finance

The numbers in this table are undoubtedly low due to the failure to include at least four major groups. These are the employees of educational institutions which were only partially included in the state government data as well as scientific systems analysts and programmers, agricultural employees and the self-employed, all of which are excluded from the *Area Wage Surveys.*

A crude correction for the non-inclusion of some of the scientific and engineering related personnel might be to increase the systems analyst and programmer categories by 33 percent. This adjustment is based on our earlier discussion of the difference between the "Federal Government" and the "Large Establishment" group on Table XI. In addition, we might, somewhat arbitrarily, increase all occupations by 15 percent to correct for the other omissions. Our justifications for this last correction are that at least 10 percent of the labor force is self-employed or engaged in agriculture and that the universities and colleges are known to employ many computer people. Using these corrections we obtain Table XV which represents our best estimate of total U.S. employment in the four categories.

TABLE XIII—Estimated Staffing Ratios by CPU Size

Occupation	British	Manufacturer		
		Small CPU	Medium CPU	Large CPU
System Analyst	1.0	1.0	1.0	1.0
Programmer	1.5	2.0	1.3	1.6
Computer Operator	1.3	3.0	1.3	.5

A comparison for this estimate can be derived from the 1971 EDP salary survey of *Business Automation* which reports that there are 13.4 systems analysts and programmers per installation in the U.S. Combining this with an International Data Corporation report that there are 34,700 installations in the U.S.[22] yields an estimate that there are 465,000 systems analysts and programmers as compared with our estimate of 360,000.

TABLE XIV—Estimate of Total Computer Personnel Employed by Federal, State and Local Governments and Non-agricultural Establishments

Occupation	Employed*	Per 1000 covered employees	Ratio to systems analyst
Business Systems Analyst	97,000	1.4	1.0
Business Programmer	137,000	1.9	1.4
Computer Operator	173,000	2.5	1.8
Keypunch Operator	384,000	5.4	4.0
Total of Above	791,000	11.2	8.0

* Includes *scientific* and *engineering* personnel employed by government.

TABLE XV—Final Estimate of Total U.S. Employment of Four Categories of Computer Personnel for 1970

Occupation	Employed
Systems Analyst	150,000
Programmer	210,000
Computer Operator	200,000
Keypunch Operator	440,000
Grand Total	1,000,000

Since criticisms can be levelled against both estimating procedures, the discrepancy is probably not surprising.

Our estimates are remarkably close to those made by the BLS in their *Occupational Outlook Handbook*. Their estimates are based on the average staffing per particular size of CPU, the numbers of the various size CPU's in use, and correction factors for productivity changes. This approach which is entirely different from ours yielded estimates of 150,000 systems analysts, 175,000 programmers and 175,000 computer operators for 1968.

CONCLUSION

Despite the collection and publication of much information by various private and governmental agencies, it is still impossible to make an accurate estimate of the number of computer personnel as defined in this paper (or by any other definition for that matter). Certainly, it is in excess of the 800,000 indicated on Table XIV. Most likely it exceeds one million. Due to the lack of use of standard occupational classifications, any attempt at disagregation into occupational specialities may really be more a matter of speculation than enumeration.

Since the major objective of collection and publication of these data is to provide information for manpower planning (i.e., the matching of the numbers of persons trained to those needed for a particular occupation), the lack of detailed estimates is a serious problem. The result so far has been a period of serious manpower shortages from about 1960 to 1969 followed by the present condition of moderate oversupply. Further, present and future prospects are for large oversupplies in some occupational specialties while at the same time there are critical shortages in others.

It is clear that total employment of computer operators, programmers and systems analysts is not significantly greater than 560,000. In our companion paper we found that the current annual production of people trained for these jobs is over 170,000. To employ these would require a new entry rate of about 30 percent which is much higher than the estimates of relative openings given by BLS of about 11 percent for computer operators, 13 percent for programmers, and 18 percent for systems analysts.[23] Thus, current production trends are not commensurate with current employment needs.

Our major conclusion is that a serious potential imbalance exists and that much additional effort is needed in gathering and publishing reliable statistics in order to prevent future such imbalances.

ACKNOWLEDGMENTS

The authors are very appreciative of the many individuals, both in industry and government, who have helped them track down sources of data.

REFERENCES

1 B GILCHRIST R E WEBER
Sources of trained computer personnel—A quantitative survey
Proceedings of the Spring Joint Computer Conference
Vol 40 1972

2 *Occupational outlook handbook*
US Department of Labor Washington DC 1970-71

3 B GILCHRIST
Manpower statistics in the information processing field
Computers and Automation Vol 18 No 10 September 1969

4 *Area wage surveys*
Bureau of Labor Statistics Washington DC 1969-71

5 Office of Management and Budget Circular No A-83
Washington DC 1967

6 *Summary of ADP activities on cost, manpower, utilization in the United States government for fiscal year 1970*
US Government Printing Office Washington DC 1971

7 *Information systems technology in state government*
NASIS Report The Council of State Governments
Lexington Ky Dec 1970

8 *Public employment in 1970*
US Department of Commerce Washington, DC, April 1971

9 *Municipal government wage survey Atlanta, Georgia, May 1970*
Bureau of Labor Statistics Atlanta Ga Dec 1970

10 *Municipal government wage survey Boston, Massachusetts, June 1970*
Bureau of Labor Statistics Boston Mass 1970

11 *Municipal government wage survey Kansas City, Missouri, May 1970*
Bureau of Labor Statistics Kansas City Mo 1970

12 *Municipal government wage survey Chicago, Illinois, May 1970*
Bureau of Labor Statistics Chicago Ill Feb 1971

13 *Municipal government wage survey Buffalo, New York, October 1970*
Bureau of Labor Statistics New York NY May 1971

14 *Municipal government wage survey New Orleans, Louisiana, May 1970*
Bureau of Labor Statistics Dallas Texas Dec 1970

15 *Municipal government salary survey Philadelphia, Pennsylvania, July 1970*
Bureau of Labor Statistics Philadelphia Pa Jan 1971

16 *Information systems for the city of New York*, Vol IV
Office of the Mayor New York City March 1969

17 *Municipal government wage survey, New York City*
Bureau of Labor Statistics New York June 1971

18 *The economy at mid-year 1971 with industry projections for 1972*
US Department of Commerce Washington DC Aug 1971

19 *Employment and earnings*
US Department of Labor Washington DC Monthly

20 J D HUMPHRIES
Future staff needs
National Computer Center Newsletter No 19 Manchester England Feb/March 1970

21 *EDP nationwide salaries report—1971*
Business Automation June 1971

22 Private Communication from Patrick McGovern President International Data Corporation Oct 1971

23 *Occupational manpower training needs*
Bureau of Labor Statistics Bulletin No 1701
Washington DC 1971

Sociological analysis of public attitudes toward computers and information files*

by RONALD E. ANDERSON

University of Minnesota
Minneapolis, Minnesota

As individuals increasingly experience a variety of contacts with computers, it becomes important to know something about the impact of these contacts and how different sectors of society feel about computerization. Such knowledge would (not only) give us insight into the source of public attitudes toward computers but would provide some basis upon which to anticipate public acceptance or rejection of computerization. These considerations should be taken into account in planning future computer applications, particularly social information systems.

Data on public perceptions of computers has been surprisingly slow in appearing. Our knowledge of such perceptions is sketchy and incomplete and measurement techniques need evaluation and refinement. The pioneering work on public attitudes toward computers was done in 1963 by Robert S. Lee with a national survey supported by IBM. The results of this study were not reported for several years. Yet until this year no one attempted another large survey devoted to public opinion and computerization.

Lee reported that the public in 1963 viewed computers from two distinct perspectives: (1) beneficial tools providing a better life for man, and (2) awesome thinking machines that threaten individuality. Lee suggests that the lack of correlation between these two views of the computer implies that a simple pro-con opinion assessment strategy would be misleading.[5,p.59]

Several small scale, exploratory research efforts have been reported during the last few years. Armor and Feinhandler conducted a study for the Harvard Program on Technology and Society investigating public perceptions of technology. Included in their survey of several communities in the Boston area were a number of questions specifically on the topic of computers. They found a pronounced social view of computers as beneficial and useful tools; yet persons simultaneously expressed fear and reservations concerning some aspects of computerization, such as data banks. This study is likely to provide important information concerning the social significance of the distinction between technology and computers.

Other miscellaneous studies have focused upon differences in belief structures among different social groups. Rosenberg and his associates at a psychiatric hospital compared attitudes of patients and hospital staff toward computers. Certain differences were found but their significance is difficult to assess because controls were lacking on crucial variables.

The scant literature on public opinion toward computerization suggests that public reaction to computer technology is best characterized as diverse, complex, and indicative of individuals' prior experiences, as well as their general orientations toward life. Public perceptions of computers are probably more analogous to attitudes on public affairs issues rather than attitudes toward specific objects like typewriters. Thus it seems particularly important to explore the interrelationship of computer conceptions with numerous other variables representing various social processes and social characteristics. To pursue this end it would be useful to begin by outlining a framework for analysis.

CONCEPTUAL FRAMEWORK

The goal of this analysis is to arrive at an improved understanding of the process of the emergence of

* The research reported herein is based in part upon survey data gathered by the Minnesota poll of the Minneapolis Tribune. I which especially to thank Mr. Robert Coursen, manager of the News Research Department, for his assistance in providing data and information.
This paper was prepared prior to the release of the AFIPS/Time Inc. Report on a national survey of attitudes towards computerization. Their findings are impressively consistent with the conclusions reported here.

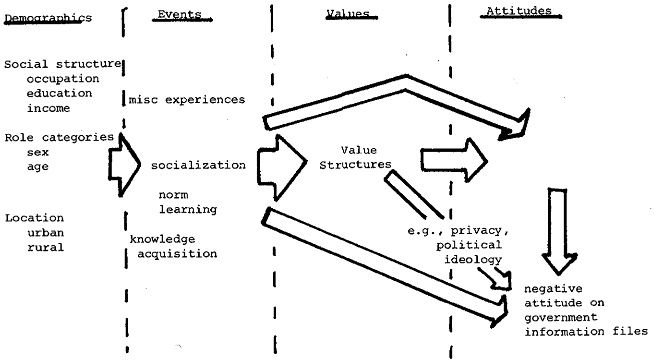


Figure 1—Conceptual scheme

particular conceptions of computer-relevant objects. Such understanding will make it possible to account for individual variation on feelings about computers and computer information files.

The orientation of our conceptual scheme is more sociological than psychological in that value orientation and experiences are stressed rather than personality factors. Furthermore, ecological-demographic characteristics are integrated with experiences and values to account for differences occurring in attitude toward computerization. Figure 1 outlines the general flow of effects generated by the combination of numerous, complexly organized variables. Demographic categories are seen to significantly determine chances of obtaining various experiences and values that in turn are crucial to an individual's cognitive process by which he arrives at particular beliefs and feelings about computerization.

Working in reverse order across the chart, each of the four sets of variables will be considered individually.

Attitudes

Two areas of affective orientation receive the focus of this investigation: attitudes about computers and attitudes about government information files. These two phenomena embody the processes defined by the other variables in the conceptual scheme. Attitude toward computers and attitude toward government information files are not entirely independent. If one has negative feelings about computers, then one is also likely to have negative feelings about government information files; these files are generally thought to be computer-based. Access to government information files typically requires computers, so if computers do not elicit good feelings, neither should the files "guarded" by them. On the other hand, computers may be negatively evaluated in part because they allow large, threatening files to survive.

While there may be a reciprocal causal relationship between attitudes toward computers and attitudes toward information files, computers chronologically preceded computer-based information systems. Thus we will assume a causal direction of computer attitude to information file attitude. Large government files existed long before computers were invented; never-the-less, widespread concern is a recent trend resulting primarily from the computer's potential for centralizing information. Attitudes toward both computers and government information files are assumed to result from a combination of (1) value orientations, (2) previous events and experiences, and (3) other attitudes.

Values

Although the conventional distinction between attitudes and values is not entirely precise, general usage restricts values to mental constructs with terminal (end-state) attributes and moral (highly desirable or ethical) implications. Attitudes are typically directed toward specific objects which are historically dependent. Given these definitions, cognitive orientations directed toward computers will be termed attitudes whereas orientations toward technological progress will be considered values. Likewise feelings about man-computer relationships are termed attitudes but feelings and beliefs on friendship, sharing, love, respect, harmony, expressiveness, and communication are thought of as values.

Not only has the values concept persisted through the history of social thought, but discussions of technology have generally been preoccupied with value issues. Recent protest movements and numerous writers have expounded anti-technological points of view. Their statements inevitably are based upon value preferences such as personal expressiveness, creativity, spontaneity, friendship, etc. On the other hand pro-technological statements are based upon value preferences such as economic prosperity, orderliness, control, etc.

The level and structure of one's values largely determines how one views the significance of technology. But technology in general is not typically an issue with which individuals are personally involved. Individuals are involved with specific technologies, specific machines. Complex value structures may not be as closely tied to specific objects as to technology in general; however, there is evidence that computers and technology are closely associated in most persons' conceptualizations. Because of its tremendous power and awesomeness, the computer has become a symbol of super-technology. The folklore, as illustrated in cartoons, science fiction, and popular science, presents this image. Computers simultaneously represent the best and the worst in technology.

Value systems other than technology-oriented value structures are likely to determine attitudes relevant to computers. The importance one places upon privacy relative to governmental or organizational needs will automatically affect one's feelings about whether or not databanks are threatening. Valuing privacy might result in the opinion that computers should not be used in an information system because computers would allow for cross-referencing of information. Political ideology also affects attitudes toward databanks. An ideology directed against strong governmental control would be wary of databank systems because of their implications for increased centralized control.

Events

The emergence of personal values and value systems is based upon a sequence of events which provide knowledge and experience to the social actor. This event series is commonly called socialization or cultural learning. It is this process that allows group norms and other social conditions to generate an impact upon individuals. Some events, tremendously more significant than others, cause immediate impact upon attitudes and values. It is this possibility that accounts for the arrows in Figure 1 going directly from "events" to "attitudes" bypassing "values." A single traumatic encounter such as a major foulup in computer dating or a computerized billing hangup might be sufficient to cause change in feelings about the desirability of computerization.

Demographics

Demographic variables such as occupation, sex, and age serve as gates opening or closing certain event sequences. The end result is a variety of life styles, role patterns and values. Lee reports a regression analysis using demographic and personality variables to predict position on his "awesome machine" attitude factor. He found that although there were correlations between education, sex, and the attitude factor, their contribution was negligible in predicting attitude on the "awesome machine" dimension. While this is superficially consistent with our conceptual scheme, ignoring the indirect effects of demographic variables is misleading. Demographic-ecological factors provide the foundation for crucial experiences which in turn create different attitudes and values. Neglecting the relationships between demographic variables and attitudes toward computers because such variables are, on the surface, weak is conceptually misguided and ignores interesting relationships that will be investigated in the following sections.

DATA COLLECTION

The Minnesota Poll interviewed a randomly selected cross-section of Minnesotans age eighteen and older during July 1971. This was one of *Minneapolis Tribune*'s periodic surveys of 600 persons; included in the July 1971 survey were several questions pertaining to computerization. One of the first questions relevant to computers was the following:

Question 1. Some people say that the relationship between businesses and their customers has become too impersonal because of the computer. Are you inclined to agree or disagree?

Two-thirds (66 percent) of the interviewees agreed with this statement linking impersonal business with the computer; 28 percent disagreed; and 6 percent gave no opinion. Respondents were next asked the following question:

Question 2. Have you ever had a mistake made in a transaction that was hard to get cleared up because billing was handled by a computer?

Only about one-third answered this question affirmatively. In actuality some persons who said they had the computer-based problem may not have known for certain that a computer was involved, but the key point is that these persons *perceived* the computer responsible for the difficulty.

Whenever a respondent admitted they had experienced computer billing problems, interviewers probed into the details of the dispute. Later on the interviewers launched into questions pertaining to databanks and privacy. The first question in this series was:

Question 3. Some people say that American society is threatened by the increase in information that the government collects about individuals from the census, tax returns, social security, and so on. Are you inclined to agree or disagree?

The key element of this question is the notion of increased collection of information by the government.

Centralization of information and links between information files are implicit but not explicit; the question suggests an environment that lacks controls over access to and centralization of information. With the term "threat" the statement expresses a rather strong statement against government information collection, so it is startling that 41 percent of all respondents agreed with this statement; 53 percent disagreed; and 5 percent gave no opinion. The next question asked was:

Question 4. Which do you think is more important for society to function properly—the government's right to know about its citizens, or the individual's right to privacy?

An overwhelming two-thirds (66 percent) indicated that privacy was more important for society; 28 percent selected the government's "right to know" over privacy; and 6 percent did not choose one or the other.

In addition to these data collected from a representative sample of voting-age residents of Minnesota (the state of Minnesota is in many ways typical of the United States population as a whole [although certain minority groups such as blacks and chicanoes are not well represented)], a questionnaire survey was carried out on a convenience sample of students at the University of Minnesota. Students in several undergraduate sociology classes were asked to fill out a questionnaire which included questions mostly concerning attitudes toward computers. A total of 100 questionnaires were received and processed during the early Fall of 1971. Included among the many questions on attitudes toward computerization were the above four Minnesota Poll questions mentioned above with identical wording in identical order. The distribution of responses from the student sample are remarkably similar to the Minnesota Poll results. Of the students, 64 percent agreed with the first question (impersonal business and computers), compared with the statewide results of 66 percent. Forty-two percent of the students had encountered a computer billing problem, compared with 35 percent statewide. On the statement concerning threat from government information collection, 50 percent of the student sample agreed, contrasted with only 41 percent agreement statewide. And 83 percent of the students chose privacy over the government's right to know; 66 percent made the same choice statewide. This difference between the results of these two studies concerning this privacy issue is not as significant as might be expected; in the statewide Minnesota Poll, 72 percent in the 18-29 age bracket and 75 percent with some college experience selected privacy over the government's right to know. In summary the response distributions from the two studies are very similar on the questions pertaining to computerization and information privacy.

CONTRASTS AMONG DEMOGRAPHIC GROUPINGS

The previously mentioned four questions dip into an important array of issues: the link between computers and impersonal business, the discomforting experiences of computer consumers, the perceived threat of government information collection and the value of privacy. Each of the questions is presented in Table A with breakdowns contrasting groupings on sex, age, location (urban/rural), and education. Income is not included because in most cases income trends were nearly identical to differences in education levels.

Question 1—*Impersonal business because of computers*

Looking first at question #1, by reading down the column of Table A it is possible to readily note that differences exist between men and women and also

TABLE A—Computerization Responses With Demographic Breakdowns—Minnesota Poll Statewide Sample, July 1971

		Quest 1	*Quest 2*	*Quest 3*	*Quest 4*
	No. of Persons	Imper-sonal	Billing	Gov't Threat	Privacy
Total	600	66*	35	41	66
Sex					
Male	292	60	37	47	70
Female	308	70	33	36	63
Age					
18-29	153	59	34	41	72
30-59	293	71	42	42	65
60+	154	64	27	38	61
Location					
Urban	425	66	40	42	67
Rural	175	63	25	42	66
Education (highest level)					
Grade (some)	119	61	19	43	64
High School (some)	297	68	33	37	62
College (some)	184	65	50	47	75

* All numbers in columns under question numbers 1, 2, 3, 4 are percentages answering the question in the indicated way.

between the younger (18-29) age group and the middle (30-59) age group. Sixty percent of the men and 60 percent of those individuals aged 18-29 agreed with the statement that linked computers with impersonal business: a greater percentage (70 percent) of the women and those respondents in the middle age group agreed with that statement. Very slight differences are found in other areas: the urban percentage exceeds the rural figure, and the high school and college educated are slightly more likely to agree with the statement than those with only a grade school education. These contrasts are based upon such small differences that they are unreliable; that is, the differences may be due to sampling variation alone. Furthermore, the difference among education levels are largely artifactual. There are many more persons in the lower education levels that did not answer the question; both agreement and disagreement are less frequent at the lowest (some grade school) education level. (This same pattern of response occurs for the income groupings as well, although the income group percentages are not included in the Table.)

To interpret the significance of these response patterns we should explore the meaning of the question. One might guess that the notion of impersonal businesses predominates and "computer" is a trivial element in the statement. This, of course, might be true with some respondents, but there is clear evidence from the student sample that the interviewees are reacting primarily to the idea of computers causing impersonality. Responses to this question were correlated with many other variables the highest correlation was found to occur with agreement to the statement, "Computerization tends to dehumanize people." In the various factor analytic investigations that we have performed on items concerned with attitudes toward computers, the negative idea of dehumanization and over-dependency have consistently turned up together in a dominant pattern. These findings suggest that agreement with the statement on impersonal business caused by computers is a fairly adequate indicator of *negative feelings toward computers.*

Given that question #1 is an adequate device to assess negative attitude toward computers, we can expand our interpretation of the data. Women are substantially more likely to have negative attitudes toward computers as are those in the middle-range (age 30-59) age groups. It would seem that in our society the different age and sex roles result in different experiences or lack of experiences such that different views of computers are acquired. Perhaps one of these experiences is frustrating encounters with computerized bills.

TABLE B

Question 1. (Impersonal . . . computer) by question 2 (billing problem) for statewide sample

Question 2. Computer billing problem ever encountered?

Question 1 Impersonal because of computer	Yes	No	Total
Agree	83	56	66
Disagree	16	35	27
NA	1	9	7
	100	100	100
	(212)	(388)	(600)*

* Total number of respondents are given in parentheses. All other numbers are percentages.

Question 2—Computer billing problem ever encountered?

Returning again to Table A, the 2nd column reveals substantial differences in susceptibility to billing problems due to computers. One is much more likely to experience a computer-based billing difficulty if one is in the 30-59 age bracket, if one lives in an urban location, and if one has a relatively high education and income level. All of these factors appear to be economic or social structural in character. That is, persons falling into the middle age, urban, and high socio-economic levels are the most likely to have money to spend; they also have greater preferences for credit systems, which automatically involve credit-users in additional billing systems. It is to be expected then that high money/credit users have an unusually high chance of encountering billing problems, especially problems due to computers.

ANALYSIS OF RELATIONSHIPS

It might be expected that persons facing frustrating experiences in clearing up computer-based bills come to view computer systems as unresponsive and rigid. This conception of unresponsive computer systems in businesses might actually grow into a long-term negative feeling about computers. To explore this possibility we examine the relationship between question #1 and question #2 (see Tables B and C). These tables show a strikingly strong association or correlation between computer-based billing difficulty experiences and tendency to agree with the notion of impersonal business resulting from computers.

TABLE C

Question 1. (Impersonal . . . computer) by question 2 (billing problem) for student sample

Question 2. Computer billing problem ever encountered?

Question 1 Impersonal because of computer	Yes	No	Total
Agree	74	57	64
Disagree	26	43	36
NA	—	—	—
	—	—	—
	100	100	100
	(42)	(58)	(101)

In both the statewide and the student samples there exists a strong tendency for those individuals who have faced a computer-based billing problem to express negative feelings toward computers. Those who have not experienced the frustration of a computer billing difficulty are much less likely to express negative feelings toward computers.

If negative experiences result in negative feelings, positive experiences may result in positive feelings. To explore this question as regards computer attitudes there must be a clarification of the kind of experiences involving exposure to computers which suggest that computers are important in creating a better life, what kind of experiences show computers as useful tools generally provide desired benefits. As yet there is insufficient data to determine whether or not class experience or job experience in computing is associated with a change in the level of negative attitude toward computers. The evidence thus far is ambiguous, which could mean that in many instances actual class or job exposure to computers is quite frustrating. There is evidence, however, that *anticipated* exposure to computers as a useful tool is correlated with a more positive attitude toward computers.

The following question was included in the student questionnaire: Do you expect to use computers in your future occupation? There were three alternatives: yes, maybe, and no. The responses, when crosstabulated with question 1 (indicating negative feeling toward computers) appear in Table D.

Although the association indicated in Table D is not unusually strong, students expecting to make occupational use of computers were less likely to agree with Question 1, then were those that did not foresee using computers in their future occupations.

Occupational use of computers is most likely to occur in the professions or highly skilled technical jobs. These occupational groups are more likely to occur in urban than in rural areas, and they are also associated with persons of higher education and income. Assuming that occupational experiences with computers are more favorable than unfavorable, we conclude that being urban, aged 30-59, having a college education, and having a high income predispose one to have positive computer experiences. This connection is graphically illustrated in Figure 2. This diagram vividly portrays the ironical fact that the same demographic characteristics pointing to positive experiences with computers also increases the chances of negative computer experiences, e.g., computer billing problems. The total effect of these opposite forces is to wash out the effects of certain demographic factors upon negative attitudes toward computers.

Those demographic variables whose effect is not largely cancelled are sex and age categories. Men and the younger age group both tend toward more positive attitudes; this is probably due to a link between these characteristics and positive occupational or educational experiences with computers. Being a woman or being over age 60 increases one's chances of having negative computer attitudes, but not having computer billing problems. In Figure 2 these two demographic groups are routed separately because these groups are probably characterized by lack of contact with computers. Since computers are culturally defined as "awesome thinking machines", it would not be surprising to discover that lack of contact and knowledge generates fear and distrust of computers, which in turn tends to lead one toward negative attitudes.

TABLE D

Question 1 (impersonal ... computer) by Anticipated occupational use of computers.

Do you expect to use computers in your future occupation?

Question 1.	YES	MAYBE	NO	TOTAL
AGREE	61	57	75	64
DISAGREE	39	43	25	36
	—	—	—	—
	100	100	100	100
	(18)	(47)	(34)	(100)

Question 3—Governmental information files and privacy

The question of whether or not a threat exists from governmental collection of information from "the census, tax returns, social security, and so on," follows the demographic patterns of Question 4 (privacy) more closely than Question 1 (negative attitude about computer). Contrasts between demographic categories are interesting only for the sex and socio-economic status (education and income) variables. Men are substantially more likely to perceive a threat from governmental information systems than are women. This difference parallels the differences between the sexes on Question 4 (privacy), while it reverses the direction of contrast on Question 1. There appears to be a distinct aspect of the female sex role that results in more negative feelings about computers; but males are more negative about information files and more likely to value privacy.

One possible interpretation of emphasis upon privacy among men is their wider variety of interpersonal and instrumental activities. The complexity and conflict of these varied activities can sometimes be reduced by isolation and privacy. Women in our culture are not generally allowed as much privacy as men. The sex role differences on attitude toward computers may be largely a result of early socialization patterns which direct girls to dolls and boys to mechanical toys. Girls are also discouraged from pursuing mathematical and technical skills and thus have more justification for viewing the computer as an "awesome thinking machine."

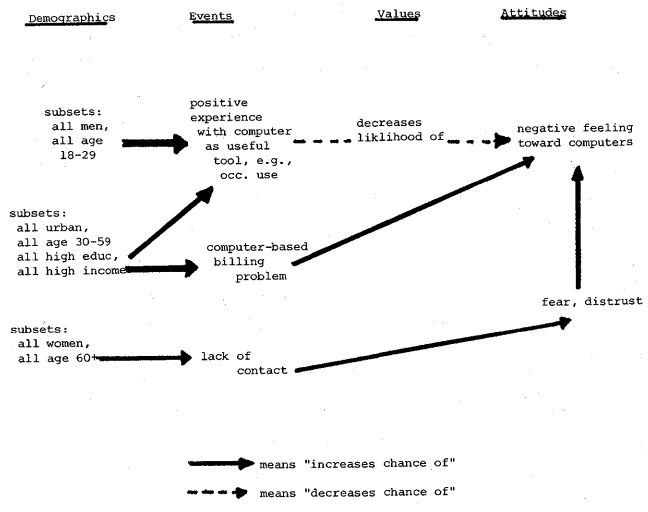


Figure 2—Interrelationships among variables pertaining to attitudes toward computers

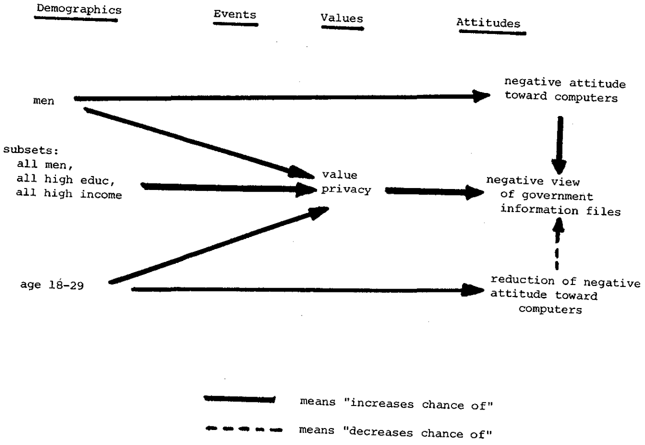


Figure 3—Interrelationships among variables pertaining to attitude toward government information files

Question 4—Value privacy or government?

The tendency to value privacy is greater among men, the 18-29 age group, the college educated, the high income groups, and the political independents. Except for the age trend, all these demographic patterns are identical to responses to Question 3.

In regard to socio-economic status, those with at least some college education and those with higher incomes are more likely to view government information collection as threatening and to value privacy.

Looking at the question from an alternative perspective, the contrast can be interpreted to result from political ideology differences or other value system differences. In this instance negative attitudes do not seem to result from events so much as values. Actually the lower income and education classes are more likely to experience negative events in connection with government information files. Databanks, especially at the local level, increasingly incorporate welfare records, arrest records, and other information more likely to involve persons of low social class. Ironically, persons in these lower socio-economic levels are the least likely to value privacy, yet they are the most likely victims of privacy invasions.

Analysis of relationships

Figure 3 presents the proposed causal relationships given the demographic findings. Again demographics are viewed as shaping the nature of values (privacy in particular) which in turn determine the nature of the attitudes toward government information files. Com-

TABLE E—Negative Attitude Toward Computer By Perceived Government Information Threat For Student Sample

Question 3. Government Information Threat

Question 1		Yes	No	Total
Impersonal Computer	Yes	74	54	65
	No	26	46	35
	Total	100 (50)	100 (50)	100 (100)

plexity in these relationships is seen to arise from the inter-relationship between attitude toward computers and attitude toward government information files. Tables E and F present the data from the Minnesota student sample for the correlation of Question 1 with Questions 3 and 4. These findings show a sizable association between responses to these questions. As Figure 3 illustrates, age and sex may indirectly affect attitude toward government information files: the 18-29 age group tends to lead into reduced negative attitudes toward computers, and men are more likely to have negative attitudes toward computers. The resulting effect of these two conditions may be to cancel out some of the variation in attitude toward information files that is due to attitudes toward computers.

IMPLICATIONS FOR CHANGE AND THE FUTURE

Cross-sectional survey data gathered at one point in time provides very little basis for generalizing to trends occurring over time. In addition, the survey approach provides little understanding of processes involving event sequences. Consequently it is imperative for future research to move toward approaches with greater control such as laboratory experiments. In addition, planning should begin toward establishing periodic assessments of public opinion using socio-computer indicators so that changes can be examined across time. Without such data there will be no way to systematically assess the impact of innovations in computer technology upon society.

TABLE F—Attitude on Computer by Value—Privacy of the Government (Question 4) for Student Sample

Question 4.

		Privacy	Gov.	Total
Question 1 Impersonal Computer	Yes	63	56	63
	No	36	44	37
		100 (80)	100 (16)	100 (96)

On the basis of age breakdowns from currently available data, it is possible to speculate about future trends assuming that the young and middle aged will simply move along into the future largely unchanged. For instance, as already noted, a larger share of those aged 18-29 value privacy over government's right to know. This trend seems to arise from developments in the youth culture and the various movements that the youth have spawned in recent years. It would appear that this emphasis upon privacy will persist and that privacy will have greater importance as a value in the population as a whole.

Establishing governmental databanks will be even more difficult in the future unless extremely secure safeguards are established to preserve individual privacy. While the young (18-29) tend to value privacy, they are not unusually concerned about the "threat" of government information collection. Members of this young group are even slightly less likely to have negative attitudes about computers.

The profile for the future suggested by these data is a populace more concerned for rigorous privacy related guarantees but simultaneously approving computer technology and related developments.

In projecting into the future, events and their impacts are the crucial uncertainty. The significance of frustrating events involving computer billing problems became evident in the earlier discussion. If the public were bombarded by a wave of frustrating computer experiences, a tremendous protest movement could easily ensue against computerization. On the other hand, if great care is taken in packaging computer-based products for the consumer, the public will gladly put up with "awesome thinking machines." The computer world still has its chance.

REFERENCES

D ARMOR S FEINHANDLER
How people perceive technology
Pp 21-24 in Sixth Annual Report 1969-1970 of the Harvard University Program on Technology and Society Cambridge Mass 1970

C E BARE
"The measurement of Attitudes toward man-machine systems"
Human Factors February 1966 71-79

R HOPPE E BERV
Measurement of attitude toward automation
Personal Psychology Vol 20 1 pp 65-70 1967

R S LEE
The computer's public image
Datamation 12 December 1966 pp 33-34, 39 reprinted in Taviss The Computer Impact 1970 Prentice-Hall pp 271-275

R S LEE
Social attitudes and the computer revolution
Public Opionion Quarterly 34 1 Spring 1970 pp 53-59

E B PARKER
Information utilities and mass communication
Pp 51-70 in H Sackman and N Nie *The Information Utility and Social Choice* APIPS Press 1970. Also reprinted in Educom Bulletin Fall pp 2-6

M ROSENBERG N REZNIKOFF et al
Attitudes of nursing students toward computers
Nursing Outlook 15 July 1967 pp 44-46
How patients feel about computers
Hospital and Community Psychiatry November 1967 pp 36-38

M REZNIKOFF C H HOLLAND et al
Attitudes toward computers among employees of a psychiatric hospital
Mental Hygiene 51 July 1967 pp 419-425

Microprogrammed significance arithmetic: A perspective and feasibility study*

by C. V. RAMAMOORTHY and M. TSUCHIYA

The University of Texas at Austin
Austin, Texas

INTRODUCTION

This study is an attempt to evaluate the feasibility of microprogrammed routines for monitoring significant digits in the numerical result of digital computers in real time. The first part is tutorial and, in the second part, microprograms for two methods of significance arithmetic are designed and evaluated.

The digital computer is a finite device that has a limit in representing numerical values; thus, in general, it maintains the most significant part of true values for computation. Therefore, errors are nearly always involved in numerical computations on a digital computer due to the finiteness of representation, truncation and round-off. Unfortunately, whether they are in the original data or generated during computation process, errors are "hidden" in the computed results. One cannot readily determine how many digits of the computed result represents the true value unless rigorous error analyses such as those by Wilkinson[1] are performed.

Significance arithmetic was suggested as a means for monitoring the significant digits in the numerical result of digital computers. If an algorithm for significance arithmetic is effectively implemented, it will provide an indication of the number of significant digits that the result retains. It will also be a powerful tool for debugging and testing numerical method for computers by tracing numerical significance at every stage of computation. Thus a study of error propagation of computational methods will be made easier. It will also pinpoint the computational stage where a large loss of significance occurs so that the computational method could be modified analytically to prevent the loss of significance and to improve the accuracy of the computed result.

SIGNIFICANCE ARITHMETICS

A number of studies have been made in the area of significance arithmetic in an effort to trace the significant digits accurately and to preserve the given number of significant digits of operands. Gray and Harrison[2] used a normalized arithmetic with an index of significance in which part of a word is used as an index to indicate explicitly the number of significant digits of the numerical value. In this method, arithmetic operations are performed in the same manner as any normalized arithmetic except an extra computation, although simple, is necessary. Metropolis and Ashenhurst[3] used unnormalized arithmetic in which numerical values are stored in an unnormalized form preserving only the significant digits. Unnormalized arithmetic is relatively simpler than other methods, but since operands in this method are not normalized, their leading zeros* must be compared with each other as well as with those of the result to adjust and preserve significance. Moore[4] proposed interval arithmetic that used two values, upper and lower bounds, to indicate the range of true value. Error bounds are computed in this method to indicate significance instead of the number of significant digits. The range for true value, i.e., the difference of the two bounds, tends to become large with a loss of significance as the number of arithmetic operations increases. Wilkinson[1] provides excellent error analyses for digital computation. His analyses, however, take a "pessimistic" approach and do not offer sharp bounds.

In this study, the first two methods are considered: namely, the normalized arithmetic with an index of significance and the unnormalized arithmetic. They are provided with simple algorithms for determining

* This work was supported by National Science Foundation Grant GJ-492 and in part by the Joint Services Electronics Program under contract AF-44620-71-C-0091.

* In Reference 3, the term "leading digits" is used since, in negative numbers, leading zeros may be represented by leading ones. In this paper, the term "leading zeros" is used because leading ones in negative number, in fact, represent "leading zeros."

the number of significant digits explicitly at each stage of computation.

The microprogrammed implementation of these algorithms are shown to be quite simple. Most importantly the computational overhead is also proved to be quite small.

DESCRIPTION OF METHODS

When the floating-point arithmetic is implemented in a computer, a normalized method is generally employed. The normalized floating-point arithmetic, however, does not provide a facility to indicate explicitly the number of significant digits in the result. It generates results with no indication as to how many of their digits might represent the true value. Since the error analysis in general is not easy, the significant digits in the result are not easily determined. Thus the idea of significant digit arithmetic or significance arithmetic is introduced.

In the significance arithmetic, the number of significant digits in the results is computed at every stage of computation and, in some cases, an adjustment is made to preserve the original significance. To realize the significance arithmetic, a number of unorthodox arithmetic methods have been proposed and experimented. Among these two methods, namely the unnormalized arithmetic by Ashenhurst[3] and the normalized arithmetic with an index of significance by Gray and Harrison,[2] were chosen for this study. In this section, the schemes used in the two methods are briefly described and illustrated with examples. The numbers in the example are represented as follows:

(exponent, mantissa) for unnormalized number;
(exponent, index, mantissa) for normalized number with index.

Significant digits of the operands are indicated by boldface types so that if an operand is **1.7320**51, for example, the first five digits **1.7320** are significant. In the example a double-length arithmetic register is assumed.

Addition

193.764 + 8453.12 = 8472.50

(a) Index method

```
(3,4, 0.193764) + (5,5, 0.845312) = (5,5, 0.847250)

                              {Exponent and index are
5,6, 0.00193764               {adjusted for decimal point
                              {aligned
+5,5, 0.845312
______________
                              {Result in double-length
  5,5, 0.84724964             {register (trailing zeros not
                              {shown)
 (5,5, 0.847250)               Rounded and smaller index is
                               used
```

(b) Unnormalized method

```
(5,0.001937) + (6, 0.084531) = (6, 0.084725)

  6, 0.0001937                {Decimal point alignment and
                              {exponent adjustment
 +6, 0.084531
______________
     0.0847247                 Result in double-length
                               register
 (6, 0.084725)                 Result rounded.
```

Subtraction

193.764 + 98.123 = 95.641

(a) Index method

```
(3,4, 0.193764) − (2,5, 0.981230) = (2,3, 0.956410)
  3,4, 0.193764               {Exponent, index adjusted for
 −3,6, 0.098123               {decimal point alignment.
______________
  3,4, 0.095641                Smaller index is used for result.
                              {Result normalized, exponent
 (2,3, 0.956410)              {and index adjusted.
```

(b) Unnormalized method

```
(5, 0.001937) − (3, 0.098123) = (5, 0.000956)
  5, 0.001937                 {Decimal point aligned;
 −5, 0.00098123               {exponent adjusted.
______________
     0.00095577                Result in double-length
                               register
 (5,   0.000956)               Unnormalized result rounded.
```

Multiplication

193.764 + 34.1526 = 6617.54

(a) Index method

```
(3,4, 0.193764) × (2,5, 0.341526) = (4,4, 0.661754)
  3,4, 0.193764
 ×2,5, 0.341526
______________
  5,4, 0.066175443864          Product in double-length
                               register; exponents added;
                               smaller index used.
 (4,4, 0.661754)               Product normalized and
                               rounded. Digit shifted into the
                               significant range (namely "4")
                               is assumed to be significant.
                               (Optimistic Approach)
```

(b) Unnormalized method

```
(5,    0.001938) × (3, 0.034153) = (6, 0.006619)
  5,    0.001938
 ×3,    0.034153
______________
  8,    0.000066188514         Product in double-length
                               register; exponents added.
 (6,    0.006619)              Significance of product is
                               adjusted to be the same as the
                               first operand that contains less
                               significant digits.
```

Division

8453.12 + 350.916 = 24.0887

(a) Index method

```
    (4,5, 0.845312) ÷ (3,4, 0.350916) = (2,4, 0.240887)
     4,5, 0.845312
    ÷3,4, 0.350916
    ----------------
     1,4, 2.408872        Smaller index is used for
                          quotient.
    (2,4, 0.240887)       Quotient normalized and
                          rounded, exponent adjusted,
                          index unchanged assuming an
                          extra digit is not good.
                          (Pessimistic Approach)
```

(b) Unnormalized method

```
    (5,   0.084531) ÷ (5,   0.003509) = (4, 0.002409)
     5,   0.084531
    ÷5,   0.003509
    ----------------
     0,  24.089769        Difference of exponents and
                          quotient.
    (4,   0.002409)       Significance of quotient is
                          adjusted to be the same as that
                          of the divisor (that contains
                          less significant digits).
```

As intuitively seen in the examples, both methods preserve and indicate the same amount of significance. It can be shown that both methods are in fact equivalent methods that use different number representations. Both methods use a carry out of the highest digit and a number of normalizing shifts (or equivalently a number of leading zeros in case of unnormalized arithmetic) as criteria for determining the number of significant digits for the result. Since they use the same criteria for determining an amount of significance, it may be concluded that they yield the same result.

A case for fully significant operands

A special case is where both operands have full significance, i.e., the value of operands happens to be precisely representable within the limit of the exponent-and-mantissa representation. If this is the case, the significance computation changes. Under the significance computation rule, a large gain in the number of significance is possible. Given two operands $A = (e_A, s_A, m_A)$ and $B = (e_B, s_B, m_B)$ where e_A and e_B are exponents, s_A and s_B significance, m_A and m_B mantissas of A and B, respectively, the modified rule is as follows:

addition/subtraction	$s_A + \lvert e_A - e_B \rvert$;
multiplication/division	$s_A + s_B$.

Clearly, in order to keep the result with increased significance a double-length mantissa is necessary. If it is not available, the result is rounded and its significance is reduced.

Unfortunately the increased significance in a double-length mantissa is possibly reduced to single-length significance by rounding, etc., in a subsequent computation. If a result with increased significance represented by a double-length mantissa is used for a computation with an operand with a single-length mantissa, the result will have the significance at most that of the latter operand, i.e., the significance is reduced to the single-length. Thus the implementation of the rule does not necessarily guarantee a result with higher significance.

Another problem associated with the special case is the detection of operands that represent a value exactly. Since the rule applies only when both operands have full significance, i.e., when they represent exact values, operands must be checked for their full significance at every arithmetic operation if the rule is to be applied. The test for full significance can cost a few steps and, if the test is performed on every operand used in arithmetic operations, it can be expensive in terms of computation time. Moreover, the probability of computing two operands with full significance is conceivably very low mainly due to rounding and truncation operated on intermediate results. The inclusion of the algorithm for the special case, therefore, will reduce computational efficiency of significance arithmetic and may not be practical unless a larger number of significance is highly desirable.

MICROPROGRAMMED SIGNIFICANCE ARITHMETIC: AN IMPLEMENTATION

The following is a brief description of microprograms for arithmetics in the unnormalized method and the normalized method with an index of significance. In this paper microprograms for addition and multiplications are considered and described. Division may be considered a multiplication in "reversed" order; its flow diagrams are presented. Many results observed in multiplication also apply to division.

The following assumptions are made in designing microprograms for significance arithmetic:

(1) A double-length arithmetic register that performs addition/subtraction in two clock cycles (the register for higher order part of operand will be called the upper accumulator, and the other, the lower accumulator);
(2) One's complement addition;
(3) One's complement representation for negative numbers.

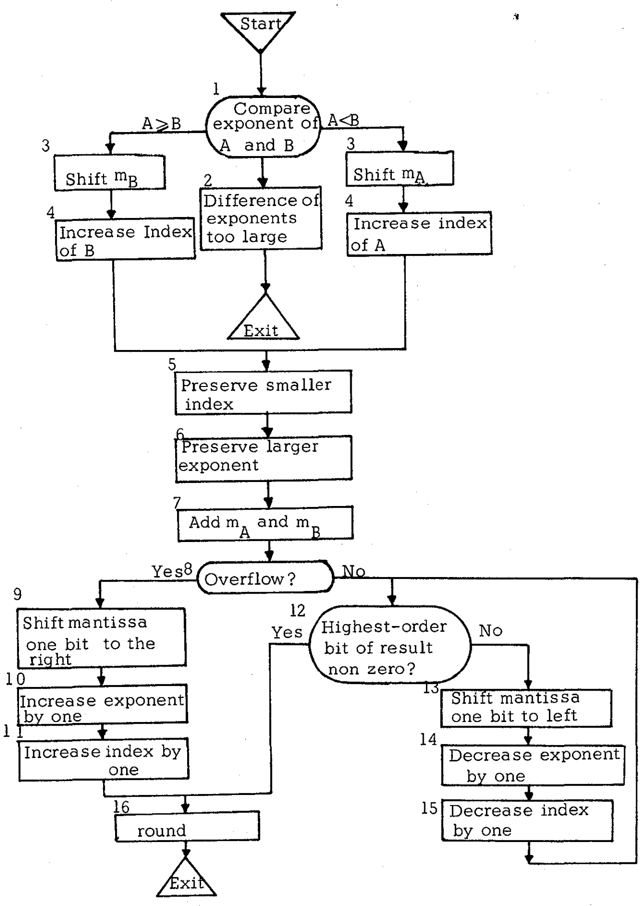


Figure 1—Normalized addition/subtraction with an index of significance

In the subsequent discussion, numbers in parentheses refer to the box bearing the same number in the flow diagram. Two operands are referred to variables "A" and "B" for convenience. The mantissas of the two operands are denoted by m_A and m_B, and the exponents e_A and e_B, respectively.

Normalized addition/subtraction with an index of significance

Addition and subtraction are carried out in a conventional manner using ones complement (Figure 1). The exponents of the two operands are compared first (1). If the difference of the two exponents is too large, i.e., if it is greater than the number of bits in mantissa, then the smaller operand is too small to have any effect as far as the mantissa and significance of the sum are concerned (2). Otherwise addition/subtraction with an index of significance commences.

Addition/subtraction starts with binary-point alignment. For binary-point alignment, the exponents of the two operands are compared (3). To compare the two exponents, their difference is computed and the sign of the difference is checked. As the mantissa of the smaller operand is shifted by the value of the difference, its lower-order bits are shifted out of the upper accumulator into the higher-order part of the lower accumulator. (A double-length arithmetic register is assumed.) For every right shift effected in binary-point alignment, the index for the smaller operand is increased

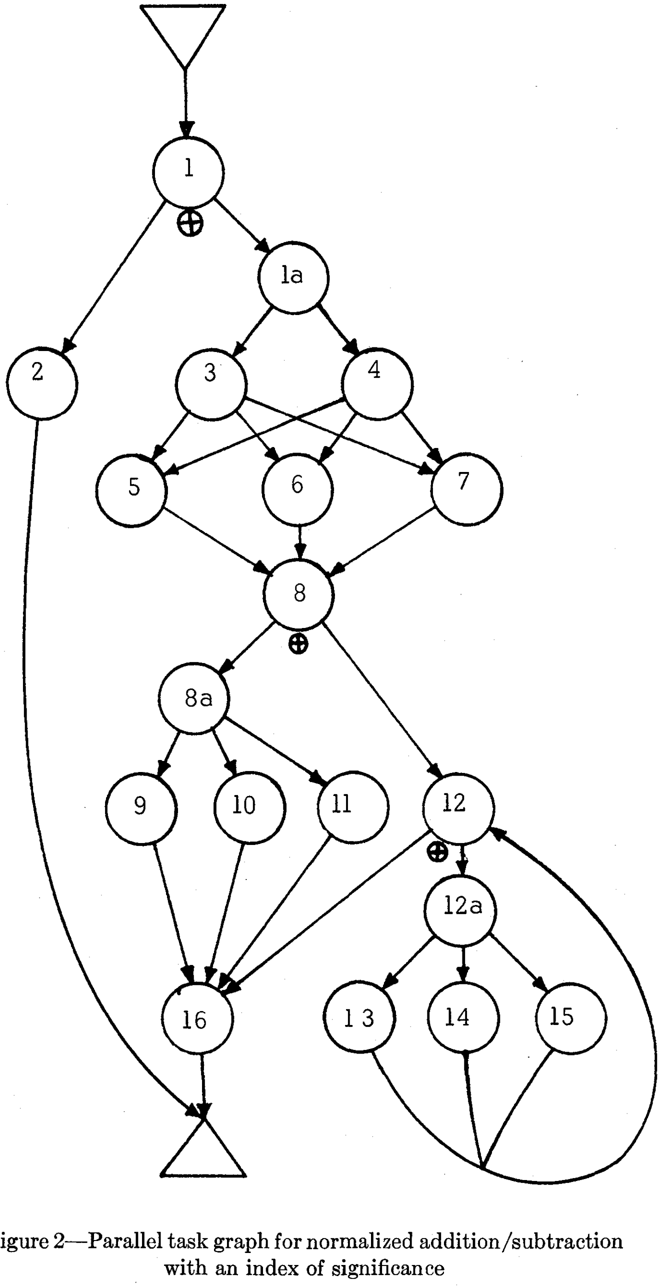


Figure 2—Parallel task graph for normalized addition/subtraction with an index of significance

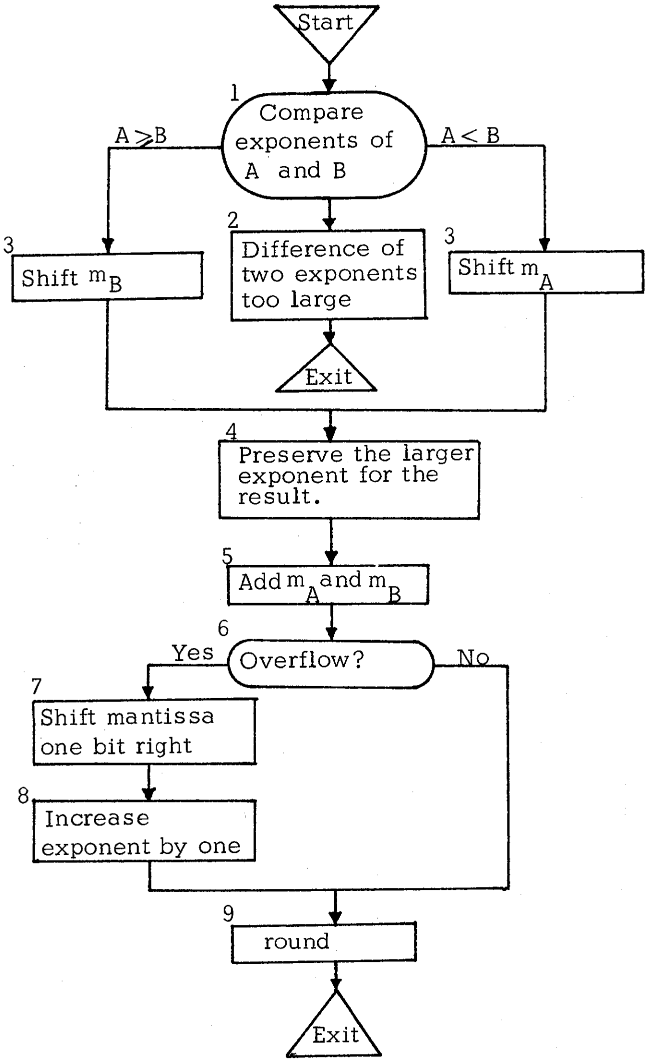
by one (4). After binary-point alignment, the indicies of the two operands are compared and the smaller is kept as the basic index for the result (5). The larger of the two exponents is preserved as the basic exponent for the result (6). After the adder is cleared, the two operands are transferred to the adder for addition (7), and an overflow condition is checked (8). If there has been an overflow, the result is shifted one bit to the right while increasing the index of significance and exponent by one (9, 10, 11). If there has been no overflow, the result is normalized. To normalize, the highest mantissa bit is tested against the sign bit (12); if they are equal, the mantissa is shifted one bit to the left (13); if they are not equal, the result has been normalized. For every normalizing shift, both the index of significance and exponent are decreased by one (14, 15). The normalized result is rounded for the final result (16). The rounding is accomplished by adding one to the highest-order bit of the lower accumulator.

Where appropriate registers and data buses among them are provided in the computer architecture, more concurrency in microprogram execution is realized. If micro-operations can be executed simultaneously without conflict, they can be coded into a microinstruction for concurrent executions (Figure 2). With an adequate computer organization, the microprogram in Figure 1 can be reorganized to take advantage of parallel data paths for a faster execution. Figure 2 shows the microprogram for normalized addition with an index of significance in parallel execution. Numbers in the node of Figure 2 refer to the micro-operations bearing the same number in Figure 1. Nodes 1a, 8a, and 12a are pseudo-nodes used as a convention for the parallel task graph representation of microprograms and cost no time in execution. The "exclusive-OR" sign ⊕ between two edges out of a node indicates that either one of the edges is traversed exclusively of the other. Note that steps 3 and 4 of the microprogram in Figure 2 do not interfere with each other and can be executed concurrently rather than sequentially. Similarly steps 9, 10, and 11 as well as steps 13 and 14 can be executed in parallel.

Unnormalized addition/subtraction

The unnormalized addition procedure, nearly identical to the normalized addition, does not require normalizing the final result. Note that the normalizing loop consisting of steps 12, 13, and 14 in Figure 2 is deleted in Figure 3 for unnormalized addition.

A major difference is that unnormalized addition does not require computation of significance indicies since the unnormalized mantissa represents only significant digits.



Start
1
Compare exponents of A and B
A ≥ B
A < B
2
Difference of two exponents too large
Exit
3
Shift m_B
3
Shift m_A
4
Preserve the larger exponent for the result.
5
Add m_A and m_B
6
Overflow?
Yes
No
7
Shift mantissa one bit right
8
Increase exponent by one
9
round
Exit

Figure 3—Unnormalized addition/subtraction

Another difference is the test for an insignificant operand (2). The number of mantissa bits is compared with the difference of two exponents in the normalized method; in the unnormalized method, it is compared with the sum of the difference of two exponents and the number of leading digits of the smaller operand. If the sum exceeds the number of bits in mantissa, the smaller operand is too small to have any effect on the result.

Possible parallelism in the unnormalized procedure is limited to steps 7 and 8 for adjustment of overflow (Figure 4). These steps are executed infrequently because unnormalized operands usually do not cause overflow. Although the unnormalized method lacks concurrent micro-operations, its microprogram is simpler and faster than the normalized method which requires the normalizing procedure.

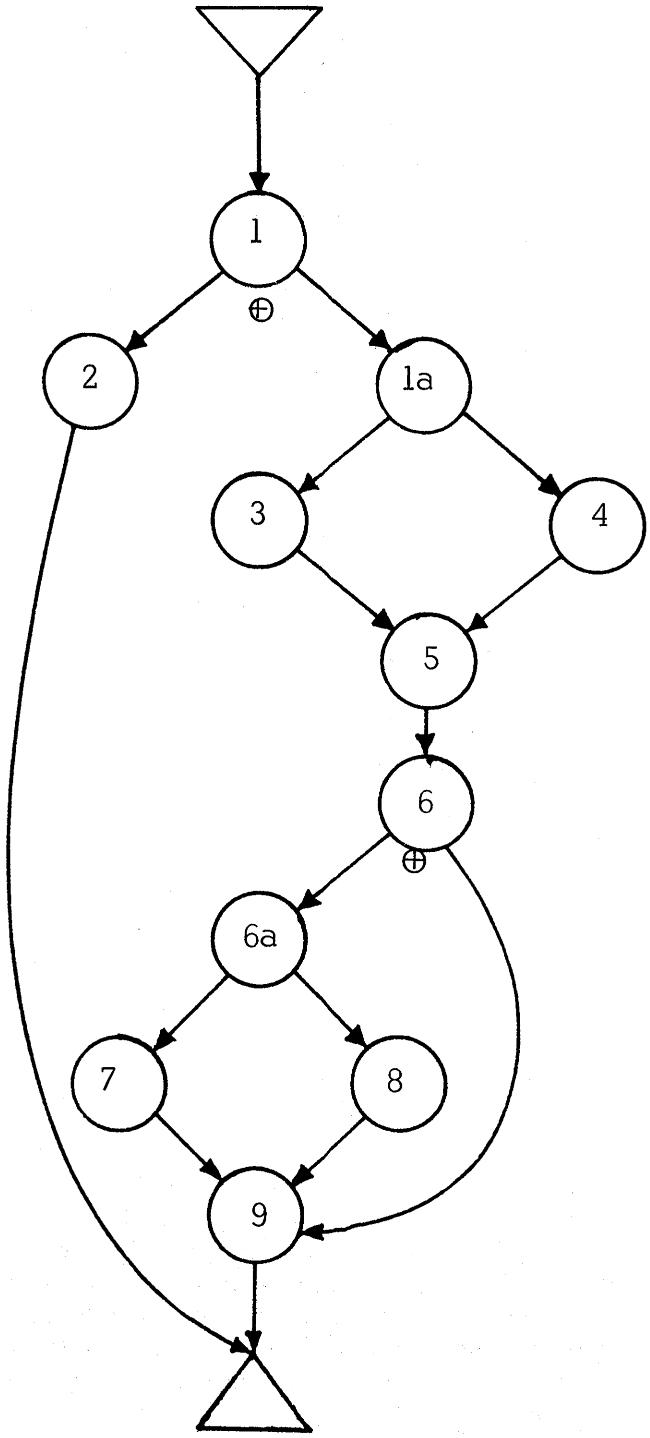


Figure 4—Parallel task graph for unnormalized addition/subtraction

Normalized multiplication with an index of significance

Normalized multiplication with an index of significance uses a conventional method: the lowest multiplier bit is examined as the multiplier is right shifted and the multiplicand is accumulated to form the product (Figure 5). The index of significance is set to the smaller of the two indicies.

The exponent of two operands are added first to form the basic exponent for the product (1). The index of

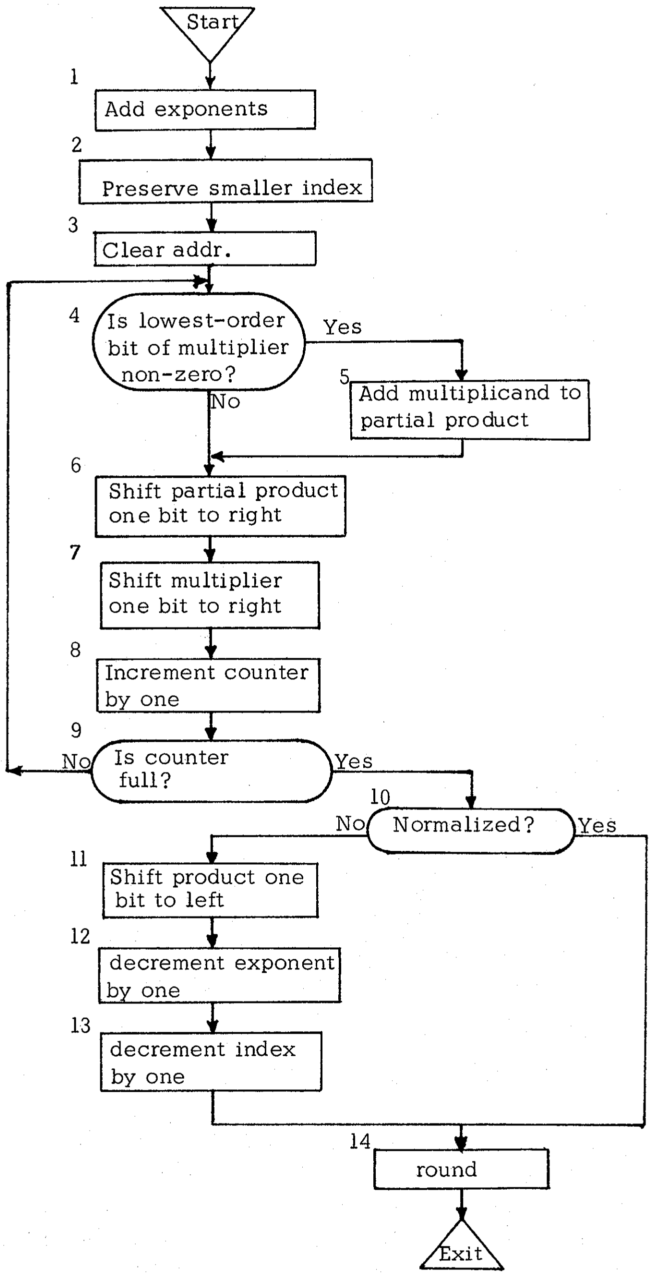

Start
1
Add exponents
2
Preserve smaller index
3
Clear addr.
4
Is lowest-order bit of multiplier non-zero?
Yes
No
5
Add multiplicand to partial product
6
Shift partial product one bit to right
7
Shift multiplier one bit to right
8
Increment counter by one
9
No
Is counter full?
Yes
10
No
Normalized?
Yes
11
Shift product one bit to left
12
decrement exponent by one
13
decrement index by one
14
round
Exit

Figure 5—Normalized multiplication with an index of significance

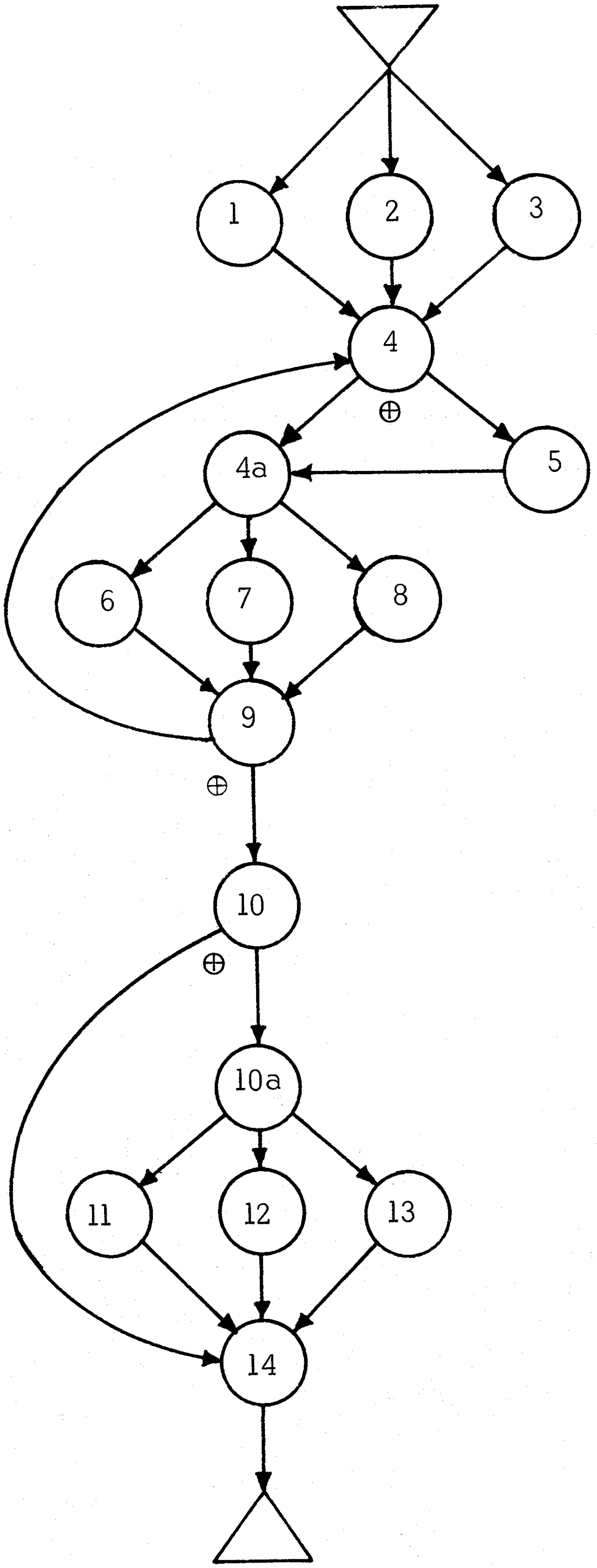


Figure 6—Parallel task graph for normalized multipl:cation with an index of significance

significance for the product is the smaller of the two (2). (See algorithm in the previous section.) The adder is cleared before the multiplication procedure commences.

The multiplier is scanned bit by bit from right to left for multiplication. If the lowest-order bit of multiplier is one, the multiplicand is added to the partial product formed in the adder; if zero, no action takes place (4, 5). The multiplier and the partial product are shifted one bit to the right for a multiplication at the next higher magnitude (6, 7). A counter counts the number of multiplier bits that have been scanned (8, 9). The result is checked for normalization (10). If a normalizing shift (at most one left shift) is necessary (11), the exponent and index are both decreased by one (12, 13). The result is rounded (14).

Clearly; steps 1, 2, and 3 are potentially parallel in the microprogram (Figure 6). Among these operations, the second that determines the smaller index requires the longest time. Similarly steps 6, 7, and 8 can be executed simultaneously. This saving is significant because these steps are loop embedded; a reduction of one step can save many cycles. For a parallel execution of steps 11, 12, and 13, at least two decrementing counters are necessary for decreasing exponent and index of significance by one.

Unnormalized multiplication

The procedure used for unnormalized multiplication is essentially the conventional multiplication procedure except the magnitude of mantissa is compared and leading zeros of the larger mantissa are counted to initialize multiplication. It takes m steps to multiply two m-bit operands. An improvement to the method suggested by Metropolis and Ashenhurst[3] by "automatically" adjusting the leading zeros in the result to that of the smaller mantissa after the multiplication procedure is completed. Where there is a carry out of the highest order significant bit (not necessarily the highest-order bit of register because of the unnormalized representation) significance is gained by one bit.

The leading zeros of the larger mantissa are counted and saved in a counter which is used later as a procedure counter for counting the number of loops executed for the multiplication proper (1). The sum of exponents is reduced by the number of leading zeros of the larger mantissa plus one (2). This subtraction enables the "automatic" adjustment of leading zeros. The subtraction by one from the sum of exponents compensates the result that is not shifted one bit to the right at the last multiplication proper loop.

Multiplication is performed in a conventional manner using shift and add operations (4-9). Note that since

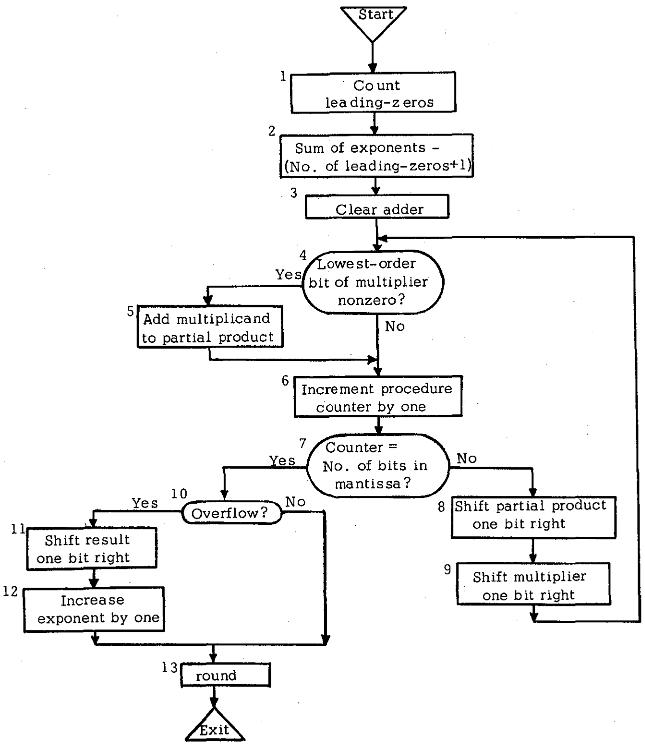

Start
1
Count leading-zeros
2
Sum of exponents - (No. of leading-zeros+1)
3
Clear adder
4
Lowest-order bit of multiplier nonzero?
Yes
No
5
Add multiplicand to partial product
6
Increment procedure counter by one
7
Counter = No. of bits in mantissa?
Yes
No
8
Shift partial product one bit right
9
Shift multiplier one bit right
10
Overflow?
Yes
No
11
Shift result one bit right
12
Increase exponent by one
13
round
Exit

Figure 7—Unnormalized multiplication

the procedure counter initially contains the number of leading zeros the multiplication loop is executed for the number of bits that represent a value in mantissa (6, 7). After the last multiplication loop is executed, the result contains either the same number of bits or one bit more to represent a value than the operand with the smaller mantissa. This is a major improvement over the original method in which a significance gain in multiplication is suppressed to preserve the number of leading zeros.

An overflow condition is checked after completing multiplication in case both operands happen to be normalized and a carry out of the highest order bit has occurred (10). If there is an overflow, the result is shifted one bit to the right and the corresponding exponent is increased by one (11, 12). A probability of the occurrence of overflow is presumably quite low, however, and these steps will be seldom executed. After an overflow condition is checked, the result is rounded to form the unnormalized product (15).

Parallelism in unnormalized multiplication may be realized in several ways (Figure 8). Steps 4 and 5 in Figure 7 are executed concurrently, or more specifically, they are constrained by step 4 execution time, and step 5 is performed at a last phase before step 4 is completed without incurring extra time. Steps 7 and 8 can be also executed concurrently when step 7 is necessary. Steps 10 and 11 can be performed concurrently if two shift registers are provided. Similarly, steps 13 and 14 can be concurrently executed. The execution time reduction by concurrency in steps 7 and 8 and steps 10 and 11 is significant because these loop steps are executed as many times as there are significant digits. A small reduction of micro-operation steps in the loop, therefore, can be a large saving in overall execution time.

EVALUATION AND COMPARISON OF METHODS

The microprogrammed algorithms for significance arithmetic are simulated on a step-by-step basis. They are compared and evaluated with regard to speed,

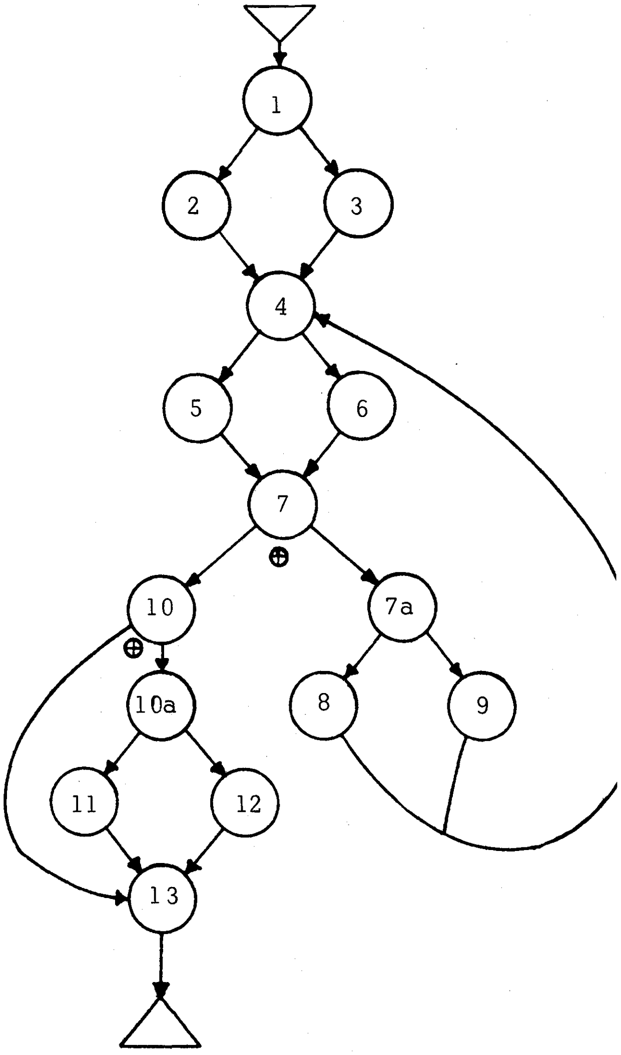
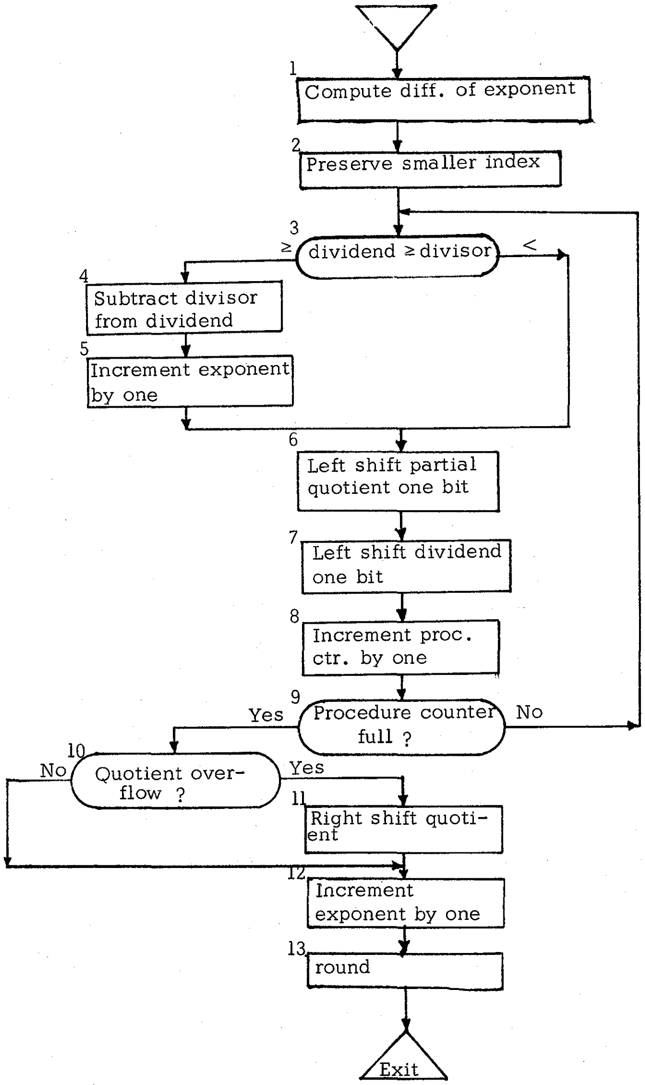


Figure 8—Parallel task graph for unnormalized multiplication

simplicity of microprogram implementation, and parallel processability. The speed of microprogram is measured in terms of the number of steps required to complete the task. The simplicity of microprogram is a capability of being represented compactly in a microprogram memory. The parallel processability is a feature that takes advantage of parallel computer structure for faster computation. The mathematical aspect of the algorithms is discussed elsewhere.[2,6]

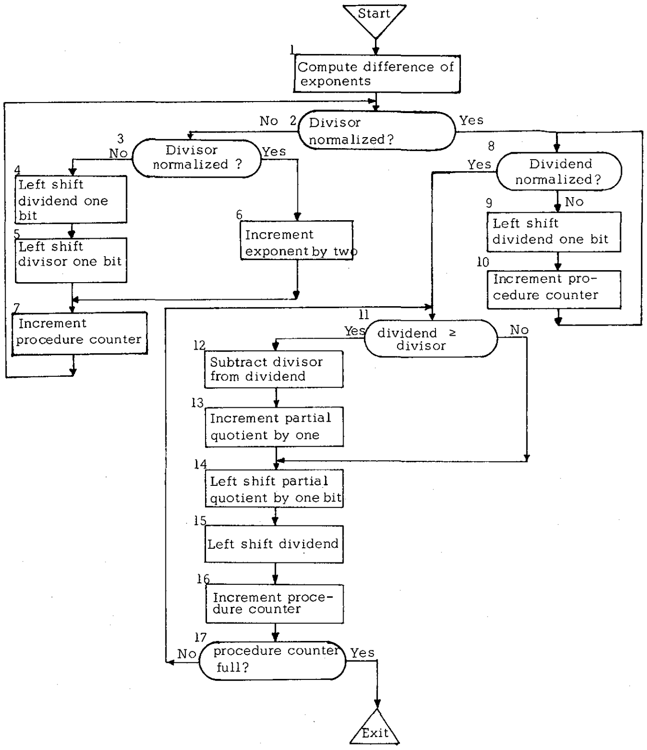
Addition/subtraction

The procedure for addition/subtraction is similar in both methods. This is readily observed in the microprogram in Figures 1 and 3. The microprogram for the unnormalized arithmetic, however, is simpler because it does not handle the index of significance. It is also faster especially when the microprograms are executed sequentially one micro-operation at a time for the same operands. This is mainly because the unnormalized arithmetic algorithm does not require the final normalization procedure. Normalization which involves shifting of mantissa and adjusting exponent (and index, in case of the index method) can slow down the procedure if there are many digits to be normalized.



1
Compute diff. of exponent
2
Preserve smaller index
3
≥
dividend ≥ divisor
<
4
Subtract divisor from dividend
5
Increment exponent by one
6
Left shift partial quotient one bit
7
Left shift dividend one bit
8
Increment proc. ctr. by one
9
Yes
Procedure counter full ?
No
10
No
Quotient overflow ?
Yes
11
Right shift quotient
12
Increment exponent by one
13
round
Exit

Figure 9—Normalized division with an index of significance



Start
1
Compute difference of exponents
No
2
Divisor normalized?
Yes
3
No
Divisor normalized ?
Yes
4
Left shift dividend one bit
5
Left shift divisor one bit
6
Increment exponent by two
7
Increment procedure counter
8
Yes
Dividend normalized?
9
No
Left shift dividend one bit
10
Increment procedure counter
11
Yes
dividend ≥ divisor
No
12
Subtract divisor from dividend
13
Increment partial quotient by one
14
Left shift partial quotient by one bit
15
Left shift dividend
16
Increment procedure counter
17
No
procedure counter full?
Yes
Exit

Figure 10—Unnormalized division

Another reason that the unnormalized method is faster is that it seldom generates mantissa overflow because of the unnormalized representation. A carry out of the highest bit of the larger operand (not necessarily the highest bit of mantissa) is accomodated in the leading zero space. This implies that the steps 7 and 8 of Figure 3 for a corrective shift and adjustment of exponent for overflow will be rarely executed. The occurrence of mantissa overflow depends on the operands: more specifically, it depends on the number of significant digits kept and the sign of operands. Hence in general the frequency of the occurrence of mantissa overflow is unpredictable. The comparative advantage of the unnormalized method over the index method for mantissa overflow, therefore, cannot be easily measured or determined.

The normalized arithmetic with an index of significance is relatively slower than the unnormalized method when the addition/subtraction microprograms are executed sequentially because it must compute the index of significance. The additional steps for index computation can be costly especially when there is a large amount of change in significance as a result of the computation. Consider a subtraction of two close values, for instance, where a loss of a large number of significant digits occurs. Clearly the mantissa of the computed result needs a large number of shifts for normalization, and for every normalizing step, the index of significance must be reduced by one to account for the loss of significant bit before it is readjusted by a number of significant bits recovered by the normalization.

The same argument holds for parallel execution of microprograms except for the last instance. If micro-operations of a microprogram are arranged and executed to take full advantage of multiple data paths for the maximum concurrency of micro-operations, the index and exponent adjustments can be performed concurrently provided that two additional arithmetic registers that accommodate the index and the exponent are available. The parallelism in microprogram will, therefore, reduce the disadvantage of the index method over the unnormalized method by eliminating the sequential execution of index and exponent adjustments to improve execution speed.

Multiplication

Assuming that the multiplication procedure is carried out sequentially in the Interdata 4 on two 60-bit operands, the microprogram for the index method takes about 467 cycles while that for the unnormalized method requires about 491 cycles. The multiplication procedure used, in both microprograms is essentially the same: register shift and addition of multiplicand as required. It takes m steps for the m-bit mantissa and each of m steps in turn consists of several micro-operations: a test of bits in mantissa against the sign bit, an addition of multiplicand into a partial product in the adder, a right shift of the partial product and of the multiplier, and an increment of the procedure counter by one. An obvious reason that the unnormalized method requires more time is that the unnormalized mantissa representation has more bits than the normalized number representation with an index. If the mantissa of the unnormalized representation is the same size as the normalized representation with an index, the multiplication procedure will take about 453 cycles. This reduction of cycles is due to fewer micro-operations executed for scaling leading digits of the two mantissas.

While both methods appear to take approximately the same number of steps and hence the same amount of time, the unnormalized method can be faster if the number of significant bits of one of the operands is small. If the number of significant bits for one of the operands is 34 instead of 44, for instance, then the total number of cycles required for the unnormalized method is 441 compared to 491. This reduction results from an increase in the number of leading bits which is due to the decrease in the number of significant bits. The number of leading zeros is counted and used as the procedure counter so that the multiplication proper is performed only on the significant bits. In other words, if k is the number of significant bits then only k multiplication steps are necessary. It is evident from the microprogram of Figure 5 that the larger is the number of leading bits (or the smaller is the number of significant bits), the smaller becomes the number of cycles required because the leading-zero scaling requires fewer cycles than a multiplication step.

The discussion so far is based on the sequential execution of microprograms. What would happen, then, if the microprogram can be executed in parallel or, more specifically, if the horizontal microinstruction format is available so that more than one micro-operation can be initiated at the same time? Assuming that a sufficient number of micro-operations (in our case, four or five would be sufficient) can be specified in a microinstruction, the microprograms can possibly be rearranged to take advantage of parallelism in execution. The index method in the rearranged microprogram takes 316 cycles which is the reduction of cycles by 32 percent over the sequential microprogram. The unnormalized method in the rearranged microprogram requires 326 cycles which is a reduction of 34 percent over the sequential microprogram. The reduction rate by the parallel microprogram for both methods is nearly the same. Speed up in the method originally suggested by Ashenhurst is hindered by the extra steps for leading digits adjustment of the final result. The improved method that adjusts leading digits "automatically" eliminated the extra steps providing more reduction and better execution time.

In summary, both normalized multiplication with an index of significance and unnormalized multiplication equally benefit from parallel data paths in a computer architecture. The former can perform separate computations on exponent, index, and mantissa concurrently, and the latter can perform computations on exponent and mantissa.

When sequentially executed, unnormalized multiplication can be executed faster, especially where the number of significant digits is small since leading-zero counting requires less time than multiplying significant digits. This peculiarity is nullified by the parallel execution of multiplication loop, however, so that the significance of operands does not affect execution time.

SOFTWARE VS. MICROPROGRAM

A Fortran program for a significance arithmetic algorithm has been implemented by Bright, Colhoun, and Mallory.[5] It monitors computer arithmetic upon request by a program in execution and provides a number of significant digits retained in the computed result at every stage of computation. The initial significance value may be either specified by the user or determined by the program at compilation or execution time. Their program is sizable (approximately 1400 Fortran statements). Being a large interpretive program, it also takes considerable computing time. A computation that requires a few seconds in normal mode would take several minutes for completion in significance mode. They concede that, for a very large computation, the use of the program would be "inordinately costly."

Microprogramming appears to be a solution to the problems associated with the software implementation of significance arithmetic algorithms. It offers a compact representation of algorithms for a smaller memory space requirement. It controls internal data flow at gate-level so that redundant micro-operations for machine code execution may be eliminated for a higher performance of algorithms.

Compactness of microprogram also implies enhanced speed in execution. Micro-level redundancy that exists in machine code program is eliminated to reduce the number of micro-level steps. An elimination of one redundant microinstruction can be a saving of many clock cycles in program execution if the microinstruction is in a loop, and can yield a significant saving in overall computation cost.

Microprogramming enables fineness of control of internal data flows. In particular, where the horizontal microinstruction and appropriate data paths are available, algorithms may be implemented to take full advantage of existing parallel data paths, which is not possible in software. Although the microprogram may become somewhat complex in this case, drastic reduction in execution times accrue for frequently used routines.[6]

To prove these points, algorithms for normalized arithmetic with an index of significance and unnormalized arithmetic were programmed both in the COMPASS assembly language for the CDC 6600 computer and the microprogramming language for the Interdata 4 computer. Although the assembly language provided relatively sophisitcated operations such as floating-point arithmetic, floating-point to fixed point format conversion, and various conditional operations on many registers, the program for addition with an index of significance required more than 50 instructions and unnormalized addition approximately 30 instructions. In order for the fairness of evaluations the following modifications were assumed for the Interdata 4 computer:

1. the machine operates on 60-bit operands;
2. eight arithmetic registers;
3. eight index registers;
4. multiple-bit shift available so that any number of bits may be shifted in three cycles.

A microprogrammed emulator is defined for each machine code operation using the microprogramming language of the Interdata 4 computer. The microinstruction of Interdata 4 uses a vertical instruction format similar to a machine code and consists of primitive and simple operations such as register-to-register transfers, simple tests for a register condition, one-bit shift, etc. Most instructions are executed in one clock cycle of 400 nanoseconds; some in two clock cycles.

The assembly language programs for the two algorithms were translated into equivalent microprograms via a set of microprogrammed emulators. The emulator-translated microprogram for addition with an index of significance consisted of 113 microinstructions, and unnormalized addition 70 microinstructions. They were simulated on two operands A and B with the following characteristics:

1. $|A|>|B|$;
2. index of significance of A is greater than that of B;
3. difference of the exponents of A and B is five;
4. result requires one-bit left shift for normalization.

The execution times of the addition with an index of significance and unnormalized addition on these operands are 119 and 69 cycles, respectively. The index method requires nearly twice as much time as the unnormalized method for bookkeeping operations such as extracting exponent, index and mantissa to perform computation, shifting, testing, etc., separately on each part.

The same algorithms were hand-coded directly in the Interdata 4 microprogramming code for a comparison with the software programs. In these directly-coded microprograms, much redundancy introduced in the emulator-translated microprograms was eliminated. The index method was coded in 78 microinstructions and the unnormalized method in 51 microinstructions. For the

TABLE I—Estimated Execution Time (in Interdata 4 clock cycles)

	Normalized Arithmetic with an Index of Significance			Unnormalized arithmetic		
	Addition	Multiplication	Division	Addition	Multiplication	Division
Software	119	633	1210	69	655	1350
Sequential Microprogram	68	467	780	43	491	810
Parallel Microprogram	35	316	510	27	326	540

same operands, the microprogram for addition with an index of significance takes 68 cycles whereas that for unnormalized addition uses 43 cycles. Comparing execution times of microprogrammed algorithm with that of software programmed algorithm, reduction realized by hand-coded microprograms is remarkable. Microprogramming the algorithms, about 43 percent of execution time is saved for the index method and about 37 percent is saved for the unnormalized method. In other words, the speed of performing significance arithmetic can be increased by approximately 40 percent by directly microprogramming.

Note that the execution of the Interdata microprograms is strictly sequential. If parallel data paths at gate-level are available, a better execution time reduction rate may be expected. Assuming that the microprograms coded for the Interdata 4 can be executed in "ideal" parallel, i.e., any possible parallelism in the microprograms may be realized, the index method for the two operands would take 35 cycles and the unnormalized method 27 cycles. Reductions realized in this case are nearly 70 percent and 61 percent, respectively, over software programmed algorithms, and approximately 49 percent and 37 percent over sequential microprograms. It may be anticipated that a further comparison of the parallel microprogram with a higher level language program for these algorithms would yield more than 80 percent reduction in execution time.

The estimated execution times of the other arithmetics is shown in Table I. Since execution time varies depending upon particular operands, an amount of significance, etc., times shown are approximate. The estimated execution times of normalized arithmetic with an index of significance are based on 44-bit mantissas and those of unnormalized arithmetic are based on 48-bit mantissas.

To illustrate the consequence of execution time reduction in significance arithmetic realizable by microprogramming, the overhead execution times for $n \times n$ matrix computation are estimated. Although the execution time for arithmetic operations varies according to the value of operands, an assumption is made that the execution times computed in the simulation and shown in Table 1 are about an average.

For an $n \times n$ matrix addition, n^2 additions are necessary. When normalized addition with an index of significance is used it would take $119 \times n^2$ cycles for a software program and $68 \times n^2$ cycles for a sequential microprogram. For $n=10$, this would mean 11,700 cycles for a software program and 6,800 cycles for a microprogram. Furthermore, it is conceivable that the difference becomes much larger if a large number of bits must be shifted for normalizing. Unnormalized addition would take approximately $69 \times n^2$ cycles in a software program and $43 \times n^2$ cycles in a microprogram. For $n=10$, the software program for unnormalized matrix addition takes approximately 6,900 cycles and the microprogram 4,300 cycles.

An $n \times n$ matrix multiplication requires n^3 multiplications and $n^2(n-1)$ additions. This would mean that multiplying two $n \times n$ matrices using normalized arithmetic with an index of significance takes

$$633 \times n^3 + 119 \times n^2(n-1) \text{ cycles in software}$$

and

$$467 \times n^3 + 68 \times n^2(n-1) \text{ cycles in microprogram.}$$

Using unnormalized arithmetic, the same matrix multiplication would take

$$655 \times n^3 + 69 \times n^2(n-1) \text{ cycles in software}$$

and

$$491 \times n^3 + 43 \times n^2(n-1) \text{ cycles in microprogram.}$$

From these execution time estimates, it is readily observed that a small amount of reduction at microprogram level may be significant in the overall efficiency of computation.

REQUIREMENTS FOR COMPUTER ORGANIZATION

There are a number of desirable features in computer organization for an effective significance arithmetic. Some of them may coincide with the numerical analysts' wishes in the design of computers for numerical computations such as a large word-size, multiple-length arithmetic registers, etc. The requirements for computer organization here, however, are considered from the standpoint of microprogramming.

The first requirement in computer organization for significance arithmetic is a double-length register to preserve significance. Wilkinson shows that if only a single-length arithmetic register is available rounding errors propagates to widen error bounds and to reduce significance.[1] Besides preserving significance, a double-length register may be used for double-precision arithmetic for greater accuracy.

A small arithmetic register large enough for computing exponent and index of significance is desirable for parallel execution of microprograms. With this register available, mantissa and exponent (and index of significance) can be computed simultaneously. For a faster normalized arithmetic with an index of significance, two such arithmetic registers may be necessary: one for exponent computation, the other for index computation. They are particularly useful in normalization in the index method. Normalization calls for left shifts of mantissa and, for every shift, reduction of exponent and index values. The two registers are used for the exponent and index computations concurrently with the mantissa shift operation to reduce overall execution time.

Two or more registers capable of shifting one bit at a time are necessary. They have many uses in both normalized and unnormalized arithmetics. They may be used in multiplication for a multiplier and partial product, in division for a dividend (and partial remainder) and a partial quotient, and in normalization. At least one of them should be able to shift its content in a group of bits at a time. Such register would reduce a time requirement in binary-point alignment for addition/subtraction in which a shift of many bits is often necessary.

Three or more mask registers for extracting any part of the register content are desirable. If they can be set to extract any part of the register content, the separation of exponent, index, and mantissa for separate computation is done faster.

A high-speed, alterable microprogram memory is desirable. It permits flexible specifications of control. When a new algorithm for significance arithmetic becomes available, it may be microprogrammed to replace the old microprogrammed algorithm without hardware rewiring. Thus, the alterable microprogram memory will prevent obsolescence of a machine with a small cost for remicroprogramming control specifications.

The microinstruction format should be horizontal. The horizontal microinstruction format permits specifying simultaneous activities of multiple data paths; it permits elemental control for efficient executions. Microprogram coding scheme must be simple, however, so that basic controls can be specified easily. If the microprogram operation code is encoded as in the case of Honeywell H4200, microprogramming becomes, somewhat restricted in terms of specifying data paths. Therefore, a more general microprogramming method and supporting tools such as microprogram interpreter, simulator, etc., are desirable for easier microprogramming.

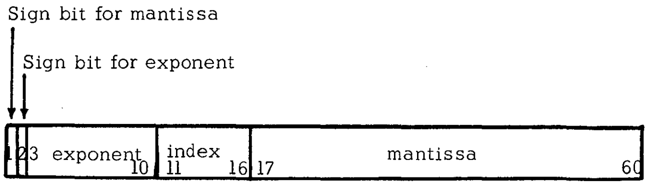


Figure 11a—Format for normalized arithmetic with an index of significance

The machine code instruction format should have a provision for specifying the number of significant digits. Basically the following machine code instructions are desirable for significance arithmetic: the input instruction for specifying a number of significant digits of input data; the significance arithmetic operation code separate from the regular arithmetic instruction; the output instruction to output the number of significant digits. The input instruction is a pseudo code that simply defines a constant and its significant digits. The significance arithmetic operation code may be the regular arithmetic instruction with its tag field set to indicate the significant arithmetic mode. The output instruction to extract and output the number of significant digits may be used to print out the number of significant digits. It may be also used to pass the number of significant digits to a routine that monitors significance and determines the next action such as a use of multiple-precision arithmetic, etc.

A possible format for a number representation in a 60-bit word for significance arithmetic with an index is shown in Figure 11a. It consists of two sign bits, one for exponent, the other for mantissa, eight bits for exponent, six bits for an index of significance, and 44 bits for mantissa. A format for the unnormalized number representation in a 60-bit word is shown in Figure 11b. It consists of a sign bit for exponent, a sign bit for

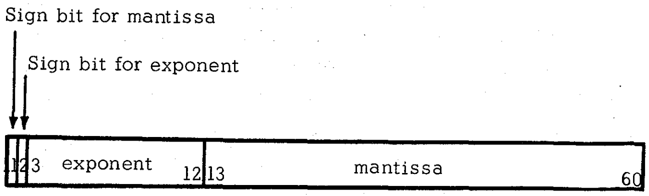


Figure 11b —Format for unnormalized arithmetic

TABLE II—Comparisons of the Two Significance Arithmetic Methods

	memory requirement	execution speed	microprogram simplicity	parallelism in procedure	range of representable numbers
Index method	larger	nearly the same	slightly complex	more parallelism	smaller
Unnormalized method	smaller	faster add/sub	simpler	less parallelism	larger

mantissa, ten bits for exponent, and 48 bits for mantissa. Clearly, for the same word size, the unnormalized format has a potential capacity for maintaining a larger number of significant digit (because of a larger mantissa) and representing a wider range of numbers (because of a larger field for exponent).

None of the requirements discussed so far is extreme or special-purpose oriented. Most of the desirable features are in fact provided by the existing, general-purpose computer except the alterable microprogram memory. A general speculation, however, is that the alterable, high-speed microprogram memory will be available at a reasonable cost in the near future with the advancement of memory technology. It will be used more widely as better microprogramming techniques are developed. In conclusion, the results suggest that the existing microprogrammable computers are reasonably well suited for microprogram implementing significance arithmetic.

CONCLUSION

Microprograms have been designed and shown in flow diagrams for the two significance arithmetic methods: normalized arithmetic with an index of significance and unnormalized arithmetic. The two methods are compared in terms of speed, simplicity in microprograms, and effects in computer organization.

Software implementation and microprogram implementation of significance arithmetic are also compared to justify the microprogrammed implementation. A software implementation of significance arithmetic appears to require a formidable amount of overhead computation.

The results of this study suggest many advantages for the microprogrammed implementation of significance arithmetic. Whereas the unnormalized method is considerably faster in addition than the index method, the index method is more suitable to a computer with parallel processable characteristics. The microprogram using computational parallelism for the index method is slightly simpler than that of the unnormalized method. Although the difference in execution time between the two methods is small, it can be significant over a long period of time especially when arithmetic functions are used more frequently in the computation.

Where memory requirements are of concern, the choice seems apparent. If the sizes of exponent and mantissa are fixed, the index method requires additional bits for the index of significance. If the mantissa consists of m bits, it requires $\lceil \log_2 m \rceil$* additional bits for the index of every operand. For an n×n matrix, this would mean that at least $\lceil \log_2 m \rceil \times n^2$ additional bits are necessary.

If on the other hand the exponent, index, and mantissa are packed into a fixed word size, the size of exponent and/or mantissa fields becomes smaller for the index number representation. The index method in this case would suffer in the range of representable numbers due to smaller exponent and mantissa fields. A smaller size for mantissa would also mean less accuracy in representing numbers.

The observations and comparisons discussed above are summarized in Table II. The entry comments in the table are for the two methods relative to each other.

In conclusion, our studies have shown beyond doubt that significance arithmetic can be implemented in microprograms for a performance that costs little overhead oveı comparable regular floating-point arithmetic.

REFERENCES

1 J H WILKINSON
Rounding errors in algebraic processes
Prentice-Hall Englewood Cliffs N J 1963

2 H L GRAY C HARRISON JR
Normalized floating-point arithmetic with an index of significance
Proc of the Eastern Joint Computer Conference
pp 244-248

3 N METROPOLIS R L ASHENHURST
Significant digit computer arithmetic
IRE Trans on Elec Computers Dec 1958 pp 265-267

* The notation $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.

4 R E MOORE
The automatic analysis and control of error in digital computation based on the use of interval numbers
Error in Digital Computation Vol 1 ed by L B Rall John Wiley & Sons Inc 1964

5 H S BRIGHT B A COLHOUN F B MALLORY
A software system for tracing numerical significance during computer program execution
AFIPS Proc of SJCC 1971 pp 387-392

6 C V RAMAMOORTHY M TSUCHIYA
A study of user-microprogrammable computers
AFIPS Proc of SJCC 1970 pp 165-181

7 R L ASHENHURST
Techniques for automatic error monitoring and control
Error in Digital Computation Vol 1 ed by L B Rall John Wiley & Sons Inc 1964

Architectural considerations of a signal processor under microprogram control

by Y. S. WU

*International Business Machines Corporation**
Gaithersburg, Maryland

INTRODUCTION

The application of microprogramming to seismic, acoustic and radar processing is well-known.[1,2,3,4] The system architecture required to address wide bandwidth signal processing problems is of a general form shown in Figure 1. In order to provide the high throughput which is required by digital signal processing, parallelism is generally accepted as the proper design concept for the signal processing arithmetic element. A microprogram processor (or a host processor) could be used to control the arithmetic element which can be either an associative processor,[5] functional memory,[6] parallel ensemble,[7] matrix array processor[8] or a vector processor.[9] The efficient design of the microprogram processor is the key to insure the high duty cycle utilization of the expensive arithmetic element hardware. Parallel architectures fall far short of their expectations because of the control problem associated with keeping all the hardware usefully occupied all the time.[10]

This paper surveys basic digital signal processing algorithms. It proposes a signal processing architecture consisting of a microprogrammed control processor (MCP), a highly efficient sequential signal processing arithmetic unit (SPAU) and necessary buffer memories. Emphasis is placed on the MCP architecture because of its importance in enhancing the system performance. An application example, optimum processing[11] in frequency domain, is given to verify the applicability of the architecture.

Hardware technology surveys, and fabrication and packaging considerations are beyond the scope of this paper. The firmware and microprogramming support software discussions are also omitted.

BASIC DIGITAL SIGNAL PROCESSING ALGORITHMS

A wealth of literature, including several very good text books is available in signal processing and digital signal processing.[12,13,14] A brief description of some of the processing algorithms is included here. In general, linear filter theory and spectrum analysis techniques form the basis of digital signal processing. Time-domain and frequency domain equivalence of processing is widely assumed. In practice, time-domain digital processing is only used in real time applications with relatively short sample data block. The fast Fourier transform (fFt) algorithm[15,16] provides the efficient digital processing link between time and frequency domains, and frequency domain processing is preferred for narrow-band analysis with fine resolution and large quantities of input data.

Time-domain processing

Consider a sampled signal $\{x(t_n)\}$, where $t_n=nT$, $n=0, 1, \ldots n$ and T is the sample period. A linear digital filter is written as:

$$y(t_n) = \sum_{m=0}^{N} h(t_m)x(t_{n-m}) \tag{1}$$

where $\{y(t_n)\}$ are filtered outputs of $\{x(t_n)\}$ and $\{h(t_n)\}$ are sampled impulse responses of the filter.

Similarly, the correlation function of $\{f(t_n)\}$ and $\{g(t_n)\}$ can be written as:

$$l(t_n) = \sum_{m=-N}^{N} f(t_{m+n})g(t_m) \tag{2}$$

Another type of time-domain processing, which is known as recursive filter, is commonly used for limited number of samples to compute successive value of out-

* Presently employed at Naval Research Laboratory, Washington, D.C.

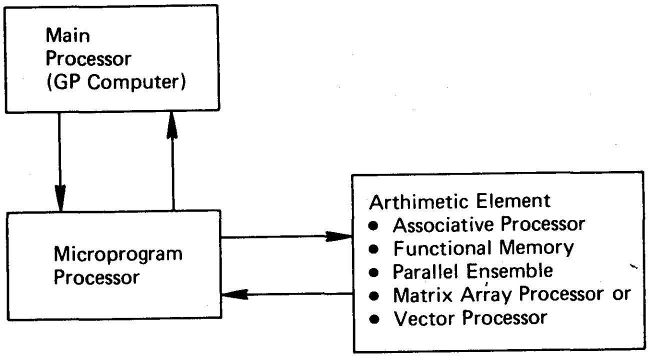


Figure 1—System organization

puts. Let $\{x(t_n)\}$ be the input signal samples, then the filtered outputs $\{y(t_n)\}$ are expressed by the linear difference equation:

$$y(t_n) = \sum_{k=1}^{N} W_k y(t_{n-k}) + \sum_{k=0}^{r} \lambda_k x(t_{n-k}) \qquad (3)$$

where $\{w_k\}$ and $\{\lambda_k\}$ are weighting or filter coefficients of the recursive filter.

For N sample points, time-domain processing requires N^2 multiply-sum operations. Therefore, real time application of time-domain processing is limited to short sample data blocks only. Typically, N equals 64 or less.

Frequency domain processing

Let $\{X(f_i)\}$ be the discrete Fourier transform of a signal $\{x(t_n)\}$ that

$$X(f_j) = \sum_{n=0}^{N-1} x(t_n) \exp(-i2\pi f_j t_n) \qquad (4)$$

where $i=\sqrt{-1}$, $j=0, 1, 2, \ldots N-1$ and $f_j=j(1/NT)$.

This discrete transform involves a couple of assumptions. Equally spaced time samples are assumed. Also the sampling rate must be above the Nyquist rate, twice the frequency of the highest frequency in the waveform being sampled. When these criteria are met, the discrete Fourier transform has parallel properties to the continuous transform. The original waveform can be completely recreated from the samples. Transformations between the time and frequency domains are performed by using the discrete transform and its inverse. It can be shown that the equivalent frequency domain digital filter in equation (1) can be written as:

$$Y(f_j) = X(f_j)H(f_j) \qquad (5)$$

where $\{Y(f_j)\}$ and $\{H(f_j)\}$ are discrete Fourier transforms of $\{y(t_n)\}$ and $\{h(t_n)\}$ respectively.

Similarly, the equivalent correlation function in frequency domain can be shown as:

$$\Phi(f_j)=\bar{F}(f_j)G(f_j) \qquad (6)$$

where $\{\bar{F}(f_j)\}$ is the complex conjugate of the discrete Fourier transform of $\{f(t_n)\}$.

As a special case, the power spectrum density function is:

$$|F(f_j)|^2=\bar{F}(f_j)F(f_j) \qquad (7)$$

It is seen that for N frequency domain samples, the equivalent digital filtering or correlation function requires N complex multiplications instead of N^2 product-sum operations in the time-domain. Tremendous computational savings for large N can be achieved if an efficient processing link between time domain and frequency domain is established. The fast Fourier transform algorithm is this missing link.

Fast fourier transform

Since 1965, a great deal of attention has been given to the *fFt* algorithm by the digital signal processing community. Interested readers can find detailed derivations and variations of the algorithm in references listed in the Bibliography.[13,15,16,17] A simple derivation is included below.

Let's rewrite the discrete Fourier transform expression shown in equation (4).

$$A_r = \sum_{k=0}^{N-1} x_k \exp(2\pi irk/N) = \sum_{k=0}^{N-1} x_k W^{rk} \qquad (8)$$

where: $i = \sqrt{-1}$

$W = \exp(2\pi i/N)$

N = number of samples

r = harmonic number $= 0, 1, \ldots, N-1$

k = time-sample number $= 0, 1, \ldots, N-1$

Thus A_r is the rth coefficient of the Fourier transform and x_k is the kth sample of the time series.

The samples, x_k, may be complex, and the coefficients, A_r, are almost always complex.

Working through an example in which $N=8$ will illustrate some of the calculation short cuts.

In this case: $j=0, 1, \ldots, 7$

$k=0, 1, \ldots, 7$

To put these into binary form:

$$j=j_2(2^2)+j_1(2^1)+j_0$$

$$k=k_2(2^2)+k_1(2^1)+k_0$$

where: $j_0, j_1, j_2, k_0, k_1, k_2=0, 1$

Thus:

$$A(j_2, j_1, j_0)$$

$$=\sum_{k_0=0}^{1}\sum_{k_1=0}^{1}\sum_{k_2=0}^{1}\chi(k_2, k_1, k_0)[W^{(j_24+j_12+j_0)(k_24+k_12+k_0)}] \quad (10)$$

Now the W can be broken down further:

$$W^{(j_24+j_12+j_0)k_24}=[W^{8(j_22+j_1)k_2}]W^{j_0k_24}$$

$$W^{(j_24+j_12+j_0)k_12}=[W^{8j_2k_1}]W^{(j_12+j_0)k_12}$$

$$W^{(j_24+j_12+j_0)k_0}=W^{(j_24+j_12+j_0)k_0} \quad (11)$$

Since $W^8=[\exp(2\pi i/8)]^8=\exp(2\pi i)=1$, the bracketed terms equal one, and can be dropped from the computation. (Note that $[\exp(2\pi i)]^2=1^2=1$.) This saves many calculations.

Then $A(j_2, j_1, j_0)$ can be obtained by sequentially calculating the xs as follows:

$$\chi_1(j_0, k_1, k_0)=\sum_{k_2=0}^{1}(k_2, k_1, k_0)W^{j_0k_24}$$

$$\chi_2(j_0, j_1, k_0)=\sum_{k_1=0}^{1}1(j_0, k_i, k_0)W^{(j_12+j_0)k_12}$$

$$\chi_3(j_0, j_1, j_2)=\sum_{k_0=0}^{1}=\chi_2(j_0, j_1, k_0)W^{(j_24+j_12+j_0)k_0}$$

$$A(j_2, j_1, j_0)=\chi_3(j_0, j_1, j_2)$$

Once these computation savings were found, one may generalize that for N point fast Fourier transform $\frac{1}{2}N\log_2N$ complex multiplications and summations are required. For the equivalent digital filter operation in equation (1), $N(1+\log_2N)$ complex multiplications are performed including the *fFt* on input samples and the inverse transform to obtain time domain filtered outputs. Comparing with N^2 operations required for (1), this is a worthwhile saving in processing load, when N is large.

What are the basic operations?

Product-sum and complex multiplications!

A PROPOSED SIGNAL PROCESSOR ORGANIZATION

The basic arithmetic operation performed by a signal processor is the high speed multiplication in the form of product-sum for time-domain processing and complex multiply for the frequency-domain computations and the *fFt* algorithm. These operations are always performed on arrays or blocks of sensor data. In other words, signal processing deals exclusively with 'structured' data. A system architect faces:

1. The design of a high speed **S**ignal **P**rocessing **A**rithmetic **U**nit (SPAU).
2. The problem of how to keep this arithmetic unit efficiently and usefully busy.

The latter poses a bigger challenge because it dictates the system through-put by collecting and controlling sensor inputs, and structuring the input data in a manner that can be most efficiently accepted by the SPAU. Typical functions are:

- I/O control
- Multiplexing or Decommutation
- Data conditioning
- Scaling
- Orthogonal addressing
- Format conversion
- Data buffering, blocking and packing.

The above listed preprocessing requirements for a SPAU are characterized by:

- Relatively simple algorithms
- Highly iterative operations
- Low precision

The advantages of using a **M**icroprogrammed **C**ontrol **P**rocessor (MCP) rather than special purpose hardware for these interface functions are the lower cost of such a general purpose architecture and the flexibility provided by the ability to change the microprogram. Furthermore, microprogramming implementation of these functions offers 5-10 times performance gain over a conventional general purpose computer of comparable technology.[1,2,3] In addition, macro signal processing functions can be provided by properly sequencing the SPAU under microprogram control. Some of these

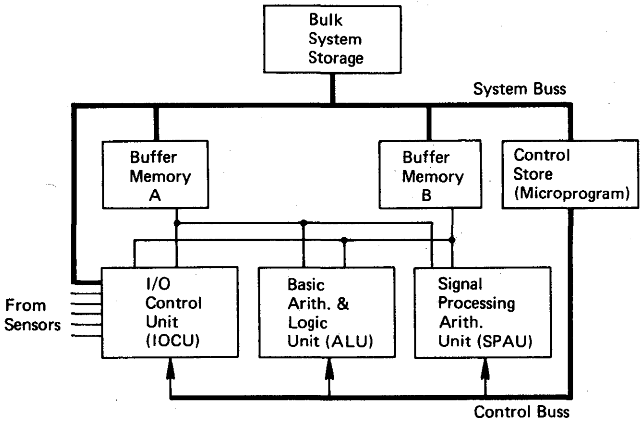


Figure 2—Functional diagram of a microprogrammed signal processor

macros could be:

- Convolution Filter
- Recursive Filter
- Beam Forming
- FFT
- Inverse FFT
- Correlations
- Power Spectrum
- Filter
- Unpack
- Matrix Operations

By requiring the system to be under microprogrammed control, the designer is permitting a single piece of hardware to be specialized to a particular type of signal processing calculation by allowing for the design of an optimum 'instruction set' for that calculation to be loaded into the control store of the MCP.

Figure 2 depicts the functional diagram of a signal processor under microprogram control. The major components are System Storage, MCP, and SPAU.

System storage hierarchy

The structured nature of signal processing requires a block (or page) of data to be operated upon by the SPAU. Therefore, SPAU performance specifications define the buffer speed requirements for each 'page' and the system through-put requirement determines the transfer rate between the system bulk store and the buffer memories. A system storage hierarchy is implied. As to the microprogrammed Control Store (CS), one may consider each signal processing kernel as a module (or page) with highly repetitive execution duty cycle. If Writable Control Store (WCS) is considered, a dynamic paging hierarchy can again be established for the microprogram execution.[18,19] Since both data and the programs are sequential and block in nature for signal processing, no cache requirement is foreseen. For relatively long data blocks, buffer paging with respect to the system bulk storage can be accomplished through the MCP I/O control unit. No additional paging hardware will be required.

Buffer memories

At least two independent buffer memories will be required because of the over taxing demands on buffer memory cycles by the pipe-lined SPAU operations while MCP ALU and IOCU are preparing the next block of data for SPAU consumption. Two buffer memories in conjunction with MCP can only support a SPAU with a 4-cycle basic operation. If a 2-cycle SPAU is required, four independent buffer memories will be needed to achieve the desired performance.

A 64-bit buffer memory interface is proposed for the purpose of increasing real time instantaneous signal input bandwidth as well as enhancing the paging interface efficiency with the bulk system storage. Experience indicated that each buffer memory should be expandable vertically in 256 by 64-bit words increments up to 4K words. 1K to 2K 64-bit words buffer size is commonly seen. It is intended that the buffer memory is the same monolithic storage used for the microprogram control store for commonality and logistic simplicity. The speed of the buffer memory is defined by the operational requirement of SPAU.

Control store

A 64-bit wide control store compatible with the buffer memory is used for the microprogram storage. Each micro-instruction is capable of executing the following operations in parallel:

- Access the Buffer Memories for One or More Operands
- Manipulate Local Registers and Arithmetic Registers
- Perform Arithmetic and Logic Functions
- Decode and Status Sensing
- Decision Making
- Form Next Micro-Instruction Address
- Other Special Controls

Allowing multiple micro-instruction formats, one can easily reduce the micro-instruction width to 32-bit with a degradation of MCP performance by less than 15 percent. Double fetch of micro-instructions can be considered; however, microprogramming will be more difficult in this case.

Since writable control store is used in the system, dynamic paging of microprograms will be considered. Tight kernels are characteristic for MCP microprograms in the signal processing environment. Small page size (i.e., 64 micro-instructions) may be adequate for a dynamic control store size of not exceeding 1K 64-bit words.

Bulk system storage

Bulk System Storage can be provided as an I/O attachment to the MCP-SPAU signal processing system in the stand alone case. Or, the signal processing subsystem can be considered as interfacing through the bulk system storage with the central computer complex. In this case, bulk system storage can be a part of the shared CPU main memory or auxiliary large core storage (LCS).

The speed requirement of the bulk system storage is dictated by the type of processing performed in the MCP-SPAU. Assume a block of data with N points are first transformed into N spectral elements in the frequency domain and then filtered by N corresponding filter coefficients; the following are observed:

Data Transfers—

Bulk System Storage to MCP-SPAU

N	Input Data Points
N	Filter Coefficients

MCP-SPAU to Bulk System Storage

N	Filtered Outputs

Computations—

$N+\frac{1}{2}N \log_2 N$	Complex Multiplications

If single cycle data transfer and 2-cycle complex multiply are assumed, the speed ratio between the bulk system storage and the buffer memories is obtained as $(2+\log_2 N)/3$. When $N=1024$, the speed ratio equals 4. This allows bulk system storage bandwidth to be 4 times slower than the buffer memory speed.

Local stores

Local storages will be provided in MCP and SPAU data flows for internal arithmetic operations. Further discussions are deferred until later sections.

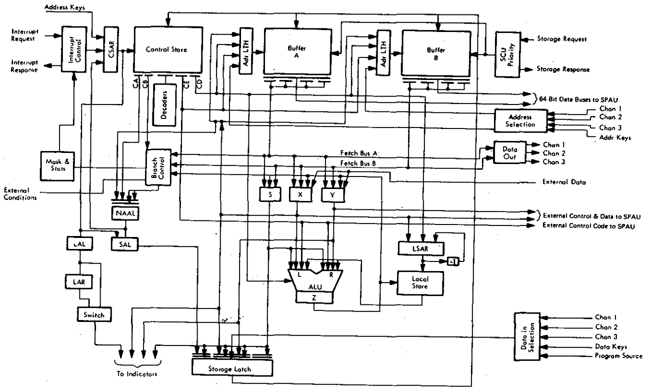


Figure 3—Microprogrammed control processor data flow (MCP)

Microprogrammed control processor (MCP) architecture

The architecture of the MCP is oriented toward its signal processing application as an interface and control processor. The salient features of the MCP required by this application are:

a. High-speed buffer and channels
b. Efficient interrupt scheme
c. Simple external interfaces
d. Ease of microprogramming

These design goals were achieved by:

a. Using two independent monolithic storage units as buffer memories and linking them with a wide 64-bit channel interface.
b. Using only three registers in the data flow and matching the storage cycle time with the internal processing time in order to permit the interrupt overhead operation (save and restore) to be performed in three micro-instructions.
c. Using a monolithic read-write storage as a microprogram control store. A 64-bit microprogram instruction provides a control format that is 5 to 10 times more powerful than typical 16-bit instructions found in minicomputers and provides the computing power necessary to perform the interface tasks. A read-write control store provides the ease of program modification required to efficiently debug the operational microprograms. It also offers dynamic paging of microprograms through the system storage hierarchy.

The basic data flow is shown in Figure 3. The functional units include:

a. Sequence Unit. The sequence unit is that portion of the machine which controls the sequence of logical operations by determining the order in which control words are addressed from the control store. The sequence unit operations are controlled by the control store word, storage bus condition, data flow conditions, machine status, and channel conditions. The address for each control store word access is assembled by the sequence unit from the above controlling sources in the next address assembly register (NAAR). The save address latch (SAL) holds assembled addresses for storage during interrupt processing. The last address register (LAR) holds the previous cycle control store address register (CSAR) when a machine stop occurs.

b. Buffer Memory Control and Bussing. Each of the basic buffer memories has a word width of 64 bits, used as four 16-bit words in the data flow. The buffer control unit (BCU) serves as an interface between the buffer memory and the various devices that use storage, such as I/O channels, the MCP data flow, and any other direct access devices such as SPAU that may be connected to it. The BCU includes a priority determination unit, an addressing unit, a storage bus, and fetch busses.

c. Data Flow. The data flow includes three 16-bit registers that provide the necessary buffering between the storage bus and the arithmetic and logic unit (ALU). They are destination registers for data fetch and ALU operations. Input selection is under direct control of the control store. All three registers have connections to the storage bus to allow them to be saved following an interrupt. Selection of operands to the ALU is controlled by a control store field. The ALU is a conventional 16-bit parallel adder and logic unit that performs a function on the selected left (L) and right (R) operands. 16-bits represents a 96 db dynamic range. Physical measurements or control signals seldom require more than 16-bit precision. Microprogrammed double-precision can be used for rare exceptions. The ALU output (Z) is latched when the function is completed and held through the late part of the control cycle, when the result is transferred to the destination register. The registers are set late in the control cycle and are held for an ALU operation in the next cycle. The ALU functions include adding, shifting, and logical operations.

d. Local Storage. A 16 word Local Storage is contained in the data flow. The local store provides 16 bit inputs to the ALU. The local store write register has an input from the ALU latch. Local store is addressed from the Local Store Address Register (LSAR). The LSAR has inputs from the X register, the CD field, and the LSAR decrementer. The Local Store can be accessed and written in one MCP cycle. It can be expanded to 32 words. Or, a second local store can be added to achieve the operations of two independent register stacks within one MCP ALU cycle.

e. Literals. Two 16-bit literals are available to the ALU for generation of constants and buffer memory addresses.

f. Interrupt Processing. Interrupt processing consists of saving program status data that would be destroyed in processing the interrupt, and restoring conditions so that the interrupted program may be continued after the interrupt has been processed. The save operation is performed in one micro-instruction:

SCXYS(O)

This micro-instruction stores the CSAR, X register, Y register, S register, stats, and masks in four consecutive locations in both buffer memories beginning at address 000_{16}. The restore operation requires two micro-instructions:

1. A(O) = X, B(1) = Y

   This loads the Y register with the vaule to be restored into S and places the CSAR value to be restored into the X register.

2. A(2) = X, B(3) = Y, Y = S, RTN

   This restores X, Y, and S registers; RTN forces CSAR to be loaded from prior X register value returning to the next address of the routine that was interrupted and also restores stats and masks.

The restore operation is interruptable. Interrupts of MCP can be generated by I/O channels, program stats and SPAU completion. There are four programmable interrupt priority levels.

Signal processing arithmetic unit (SPAU)

In preprocessing, it is observed that there is no requirement for a hardware multiplier in interface or control functions. However, an extremely high duty cycle multiplier is necessary to satisfy signal processing requirements. Furthermore, a 'structured' multiplier will be needed in this case.

In order to accommodate both time domain and frequency domain operations, the SPAU is designed to execute the following basic operation with implicit addressing:

$$D_i \leftarrow \pm A_i * B_i \pm C_i$$

where $i = 1, 2, 3, \ldots 4096$ and A, B, and C are all complex numbers

Although the SPAU provides a basic 16×16 complex multiplier, it is hardware equivalent to four 16×16 real multiplier or one 32×32 fixed point multiplier. Under the MCP microprogram control, the SPAU can function as desired in various configurations when application arises. For special case *fFt* operations, block floating point format may be assumed with 12-bit mantissa and 4-bit scaling.

The SPAU buffered operations can be interrupted by the MCP when MCP ALU registers require a single multiply operation, i.e., MCP register mode has priority over the normal buffered mode of SPAU. The SPAU buffered operations are initiated by MCP microprograms and SPAU interrupts the MCP on completion.

The SPAU design includes the necessary pipeline to assume an asynchronous operational speed. The basic SPAU operation requires two to four buffer memory cycles dependent upon number of independent buffer memories used in the system. Some special functions are included in the SPAU design such as *fFt* address generation and conjugate multiply.

The parallel matrix multiplier logic of SPAU is very straightforward. However, the amount of hardware in terms of logic gate counts is probably twice the MCP ALU/IOCU combined. It is almost anti-climactic to state again the importance of the 'lean' MCP design in order to keep the 'fat' SPAU usefully busy.

AN APPLICATION EXAMPLE—ADAPTIVE SPATIAL PROCESSING

The function flow of the application example is shown in Figure 4. The mathematical computations required to achieve the 'optimum' processing are described below.

The sensor input signals $x_K(t)$ are first transformed into spectral elements in frequency domain that

$$\{X_k(f_i)\} = FFT\{x_k(i)\} \tag{13}$$

Then the frequency domain inputs are filtered to obtain the desired beam outputs

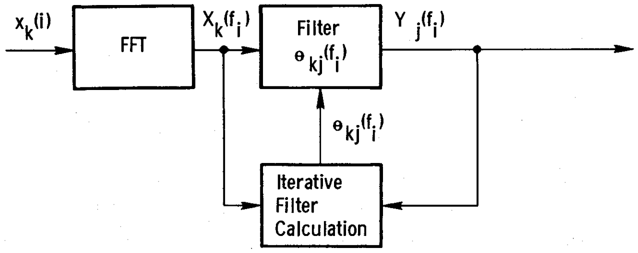


Figure 4—Adaptive spatial processing

$$Y_j(f_i) = \sum_{k=1}^{k} \theta_{kj}(f_i) X_k(f_i) \tag{14}$$

Where $Y_j(f_i)$ is the jth beam output and $\theta_{Kj}(f_i)$ are optimum filter coefficients including spatial processing in the frequency domain. These filter coefficients are updated through iterative gradient search process to minimize the output noise power. The output power spectrum is then computed

$$\{P_j(f_i)\} = \{\,|\,Y_j(f_i)\,|^2\} \tag{15}$$

Notice the block array forms of data which are efficiently processed by the SPAU. The MCP will control the paging and I/O for the system.

The optimum gradient search algorithm first computes the gradient for minimization as an input-output cross correlation function $Z_{Kj}(f_i)$ that

$$Z_{kj}(f_i) = Y_j(f_{i_0m})\{\exp(j2\pi f_i \tau_{jk}) X_k(f_{N-i}) - X_j(f_{N-i})\} \tag{16}$$

where $X_j(f_{N-i})$ are average beam outputs computed much less frequency. Notice the spatial term $\exp(j2\pi f_i \tau_{jk})$ in the equation and $j = \sqrt{-1}$ in the exponent instead of subscripted j. In order to maximize the output noise power change between mth and $(m-1)$th iteration, an iteration step size $a_j(m)$ is chosen that

$$a_j(m) = \frac{\sum_{i=1}^{N} Y_j(f_{N-i};m) Q_j(f_i;m)}{\sum_{i=1}^{N} |\,Q_j(f_i;m)\,|^2} \tag{17}$$

where

$$Q_j(f_i;m) = \sum_{k=1}^{k} X_k(f_i) \exp(j2\pi f_i \tau_{jk}) Z_{kj}(f_i) \tag{18}$$

then the iterative processing is completed by

$$\theta_{kj}(f_i;m)=\theta_{kj}(f_i;m-1)-a_j(m)Q_j(f_i;m) \qquad (19)$$

and

$$Y_j(f_i;m)=Y_j(f_i;m-1)-a_j(m)Q_j(f_i;m)$$

In addition to loop controls for the gradient search algorithm, the MCP will handle orthogonal addressing and organize the optimum filter coefficients in such a manner which can be efficiently retrieved from the bulk system storage.

The distributed processing load of this application to MCP and SPAU is normalized to MCP total processing load and tabulated below:

	Percent		
	MCP Load	4-cycle SPAU Load	2-cycle SPAU Load
FFT		9.7	4.9
Filter	9.8	69.7	35.0
Iterative Search	86.0	17.0	8.6
Power Spectrum	4.2	0.4	0.2
Total	100	96.8	48.7

It is noted that for the adaptive spatial processing 2-cycle SPAU will not be needed unless an additional MCP unit is included in the system. The 2-buffer memory configuration as indicated in Figure 2 will be applicable to this problem. The load balancing between MCP and SPAU is accomplished by the heavy involvement of MCP microprograms in the various processing loop controls of the gradient search algorithm to compute the optimum filter coefficients. For less demanding MCP control role in other signal processing applications, the 2-cycle SPAU with four buffer memories can handle higher through-put when needed. Additional MCP processing can also be applied to post detection computations which are not described in this example.

CONCLUSION

The Microprogrammed Control Processor and the Signal Processing Arithmetic Unit architecture presented in this paper blends economy, flexibility and performance in one design. It can truly be a general purpose signal processor. If ECL and bi-polar memories are used for implementation, 50 nanoseconds microinstruction time and 100 nanoseconds complex multiplication speed can be achieved for ground based applications. The hardware size in terms of logic gate counts is approximately one tenth of that of a commercially available general purpose computer with equivalent performance.

ACKNOWLEDGMENT

The author wishes to express his sincere gratitude to many people who participated in applying microprogramming to signal processing problems. In particular, he would like to thank R. G. Baron and S. G. Francisco of IBM for their pioneer work in seismic and acoustic processing, R. A. Rieth of Teledyne-Brown and J. W. Fisher of Bell Telephone Laboratories for their active involvement in radar processing and G. L. Kratz of IBM for general microprogrammed processor architecture work during the past five years.

REFERENCES

1 R G BARON W VANDERKULK S D LORENZ
A LASA signal processing system
IBM Federal Systems Division internal report Bethesda Maryland November 1965

2 *Large aperture seismic array signal processing study*
IBM Final Report—Prepared for the Advance Research Project Agency Washington D C Contract Number SD-196 July 15 1965

3 G D HORNBUCKLE E I ANCONA
The LX-1 microprocessor and its application to real time signal processing
IEEE Transactions on Computers August 1970 Vol C-19 Number 8

4 Y S WU G L KRATZ
Microprogrammed interface processor (MIP) and its application to phased array radar
Tech Note Spring Joint Computer Conference May 1971

5 J E SHORE F A POLKINGHORN JR
A fast, flexible, highly parallel associative processor
Naval Research Laboratory Report Number 6961
Washington D C November 28 1969

6 P L GARDNER
Functional memory and its microprogramming implications
IEEE Transactions on Computers July 1971 Vol C-20 Number 7

7 J A GITHENS
An associative, highly-parallel computer for radar data processing
In L C Hobbs et al eds "Parallel Processing Systems Technologies and Applications" Spartan 1970 pp 71-87

8 R S ENTNER
The advanced avionic digital computer
Ibid pp 203-214

9 *IBM 2938 array processor*
IBM System Reference Library 1967

10 J E SHORE
Second thoughts on parallel processing
To be published IEEE International Convention March 1972

11 B WIDROW P MANTEY L GRIFFITHS B GOODE
Adaptive antenna systems
IEEE Proceedings Vol 55 Number 12 December 1967

12 Y W LEE
Statistical theory of communication
Wiley N Y 1960

13 B GOLD C M RADER
Digital processing of signals
McGraw-Hill N Y 1969

14 F K KUO J F KAISER Eds
System analysis by digital computer
Wiley N Y 1967

15 G D DANIELSON C LANCZOS
Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids
J Franklin Inst Vol 233 pp 365-380 and 435-452 April 1942

16 J W COOLEY J W TUKEY
An algorithm for the machine calculation of complex Fourier series
Math of Comput Vol 19 pp 297-301 April 1965

17 W T COCHRAN et al
What is the fast Fourier transform?
IEEE Proceedings Vol 55 Number 10 October 1967

18 R W COOK M J FLYNN
System design of a dynamic microprocessor
IEEE Transactions on Computers Vol C-19 Number 3 March 1970

19 W R SMITH
Simulation of AADC system operation with an E2-B program workload
Naval Research Laboratory Report Number 7259
Washington D C April 22 1971

20 S S HUSSON
Microprogramming principles and practices
Prentice-Hall Englewood Cliffs N J 1970

A building block approach to multiprocessing

by R. L. DAVIS, S. ZUCKER and C. M. CAMPBELL

Burroughs Corporation
Paoli, Pennsylvania

INTRODUCTION

Today most computing systems have dedicated, self-contained, single processors. When the load on the system exceeds its processing capabilities, it is necessary to replace the original system with one of greater capacity. It would be far better if the original system had been modular in nature, so that additional processing modules could be added in a multiprocessing configuration with the growth in processing requirements, just as memory modules or I/O devices are added with the growth in storage or peripheral requirements. Multiprocessing systems are not new, but they have been previously limited by the processing time consumed in controlling the processor modules. With the advent of microprogrammed computers, however, control functions can now be implemented in microcode, and executed at a much faster rate than has been previously possible. In addition, microprogrammed computers are simpler and therefore more reliable than conventional computers.

This paper describes a structure for connecting and controlling a multiprocessor system using a building block technique. The hardware is modular and includes microprogrammable processors called "Interpreters", memory modules, and devices. Each Interpreter is interconnected with every memory module and every device via a data exchange network called a "Switch Interlock".

The current operating system is a simple but comprehensive control program which allows for all the basic capabilities of operating systems emphasizing multiprocessing, multiprogramming and error recovery. Plans are being developed for an extended operating system which would be a set of independent units of microcode selected from a library of such units for each individual system. The building block approach has been used by both the hardware and software implementors of the system.

MULTIPROCESSOR INTERCONNECTION

A major goal in multiprocessor system design is to increase system efficiency by the sharing of available resources in some optimal manner. The primary resource, main memory, may be more effectively shared when split into several memory "modules". A technique for reducing delays in accessing data in main memory is allowing concurrent access to different memory modules. With this concurrent access capability present, an attempt is made to assign tasks and data to memory modules so as to reduce conflicts between processors attempting to access the same memory module. Nevertheless, since some conflicts are unavoidable, a second technique (reduction of conflict resolution time) is required. These two techniques are largely a function of the multiprocessor interconnection scheme which has been discussed by Curtin[3] and others.[4,5]

Figure 1 shows three basic functional interconnection schemes. These are described in more detail in Curtin.[3]

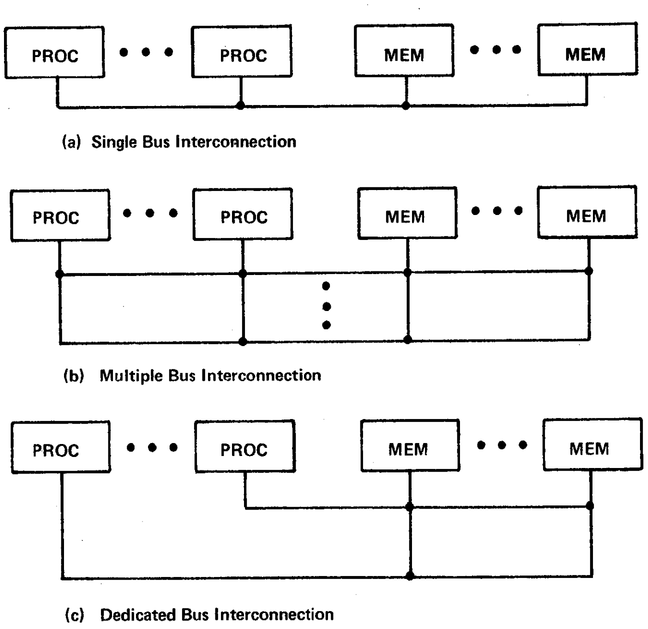
The disadvantages of the single bus approach (Figure 1) for many processors are:

(1) the obvious bottleneck in information transfer between processors and memory modules due to both bus contention and memory contention
(2) the catastrophic failure mode due to a single component failure in the bus.

A solution to the first problem has been to increase the frequency of operation of the bus.[3,6]

The multiple bus approach is merely an extension of the single bus approach where all processors contend for use of any available (non-busy) bus. The advantages are redundancy and allowing an appropriate number of buses (less than the number of processors) to handle the traffic between processors and memory modules.

The third approach utilizes a dedicated bus structure (one per processor). Although this approach requires more buses, it requires neither the logic nor, more im-



(a) Single Bus Interconnection

(b) Multiple Bus Interconnection

(c) Dedicated Bus Interconnection

Figure 1—Functional multiprocessor interconnection schemes

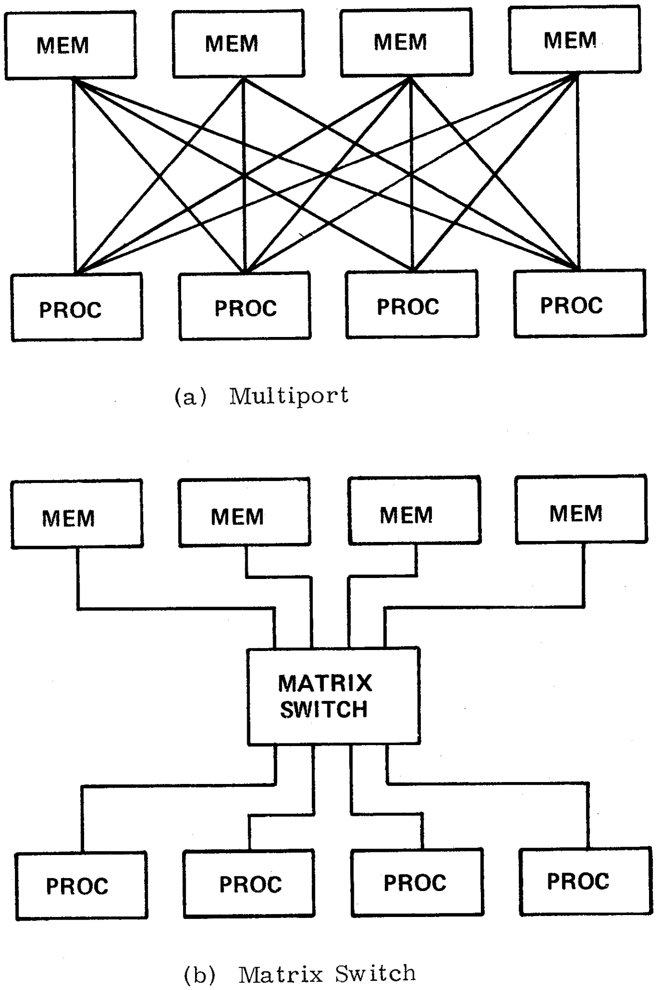
portantly, the time for resolving priority between processors requesting the use of a bus. Proponents of this approach contend that the time penalty for resolving conflicts for access to a memory module is enough of a price to pay without having to wait for the availability of a bus.

In a Hughes report,[5] the authors distinguish the physical differences between two multiprocessor interconnection schemes. The two approaches (one called multiport and the other called matrix switch) are shown in Figure 2.

The Hughes report characterizes the two connection approaches as follows:

> "In the multiport approach, the access control logic for each module is contained within that module, and intercabling is required between each processor and memory pair. Thus, the total number of interconnecting cables is the product of the number of processors and the number of memories. Each module must be designed to accommodate the maximum computer configuration.
>
> "In the matrix switch approach, the same interconnection capability is achieved by placing the access control logic for each module in a separate module. The addition of this module to the system is compensated [for] by reducing the intercables required to the sum of the processors and memories rather than the product and by not penalizing the other modules with maximum switching logic.
>
> "There generally is no speed differential between multiport and matrix arrangements. The major difference lies in the ability to grow in wiring complexity. Multiprocessors with multiport arrangements are generally wired, at production time, to the maximum purchased configuration. Future subsystem expansion generally requires depot level rewiring. This problem generally does not exist with the matrix arrangement. The maximum capacity is wired in but the switching logic complement reflects the purchased system. Subsystem expansion entails purchase of added processor/memory modules (and necessary cabinetry if required) plus the required switch matrix logic cards."

An important additional point is that even though all parts of the matrix switch are co-located, they must be designed such that the failure of one node is equivalent



(a) Multiport

(b) Matrix Switch

Figure 2—Physical multiprocessor interconnection schemes

to failure of only one element attached to the matrix switch. For example, failure of a processor-related node must not disable paths from any other processor to any memory module.

Apparent from the arguments in the Hughes report is the desire to reduce the number of wires interconnecting the processors and memory modules. A way to reduce the wiring (in addition to the use of the matrix switch) is by using serial transmission of partial words at a frequency several times that of the processors. This technique has been used by Meng[6] and Curtin.[3] The tradeoff here is between the cost of the transmitting and receiving shift registers and the extra logic necessary for timing and control of the serial transmission versus the cost of the wiring and logic for the extra interconnection nodes for a fully parallel transmission path.

Another factor adversely affecting efficiency in a multiprocessing system is a variation in the amount of computation versus I/O processing that must be done. In previous multiprocessing systems I/O functions and data processing functions have been performed in physically different hardware modules with devices being attached only to the I/O controllers (Figure 3). (This technique is typical of Burroughs D825, B 5500, or B 6700.) In the Burroughs Multi-Interpreter system,[1,2] however, processing and I/O control functions are all performed by identical Interpreters whose writable microprogram memory can be reloaded to change their function. This technique allows a configuration (Figure 4) in which the devices are attached to the same exchange as the memories and processors.

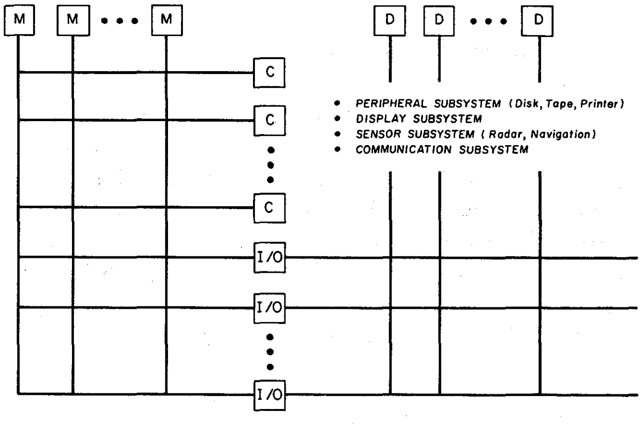

M
M
M
D
D
D
C
C
C
I/O
I/O
I/O
• PERIPHERAL SUBSYSTEM (Disk, Tape, Printer)
• DISPLAY SUBSYSTEM
• SENSOR SUBSYSTEM (Radar, Navigation)
• COMMUNICATION SUBSYSTEM

Figure 3—Centralized multiprocessor system

Switch Interlock

The Multi-Interpreter interconnection scheme for forming a multiprocessor is called a "Switch Interlock:" a dedicated bus, matrix switch with an optional amount of serial transmission.

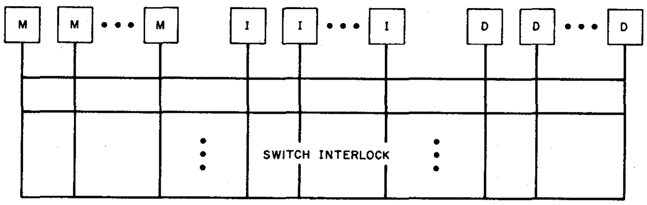


Figure 4—Distributed multiprocessing interpreter system

The Switch Interlock is a set of hardware building blocks for connecting many Interpreters to many devices and to many memory modules. Connection between Interpreters and devices is by reservation with the Interpreter having exclusive use of the (locked) device until specifically released. Connection with a memory module is for the duration of a single data word exchange, but is maintained until some other module is requested or some other Interpreter requests that module.

In any such system it is important to keep the wires and logic in the crosspoints to a minimum, while still maintaining a specified transfer rate. This is achieved by serial transmission of several partial words in parallel through the crosspoints.

Consistent with the building block philosophy of Interpreter-based systems, the Switch Interlock is partitioned to permit modular expansion for incremental numbers of Interpreters, memory modules or devices and modular selection of the amount of parallelism in the transfer of address and data through the

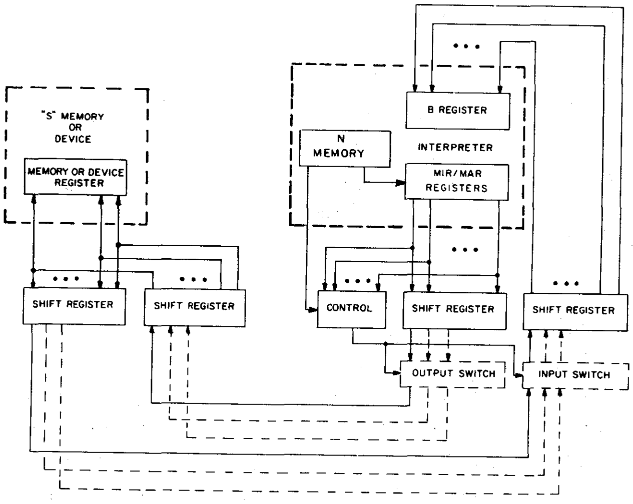


Figure 5—Implementation of switch interlock

Switch Interlock from fully parallel to fully serial. Functionally, the Switch Interlock consists of parallel-serial conversion registers for each Interpreter, input and output selection gates, parallel-serial conversion registers for each memory module and each device, and associated control logic. Figure 5 outlines the implementation of the Switch Interlock and shows the functional logic units repeated for each Interpreter, memory module, and device. The bit expandability of the Switch Interlock is shown by dashed lines between the input/output switches and the shift registers associated with the memory module, devices, and Interpreters.

The six basic Switch Interlock modules are described below:

(1) Memory/Device Controls (MDC)

The MDC is an interface between the Interpreter and the controls described below (MC and DC), and the control for the high frequency clock used for the serial transmission of data. There is one MDC per Interpreter.

(2) Memory Controls (MC)

The MC resolves conflicts between Interpreters requesting the use of the same memory module and maintains an established connection after completion of the operation until some other module is requested or some other Interpreter requests that memory module. This unit handles up to four Interpreters and up to eight memory modules. System expansion using this module may be in number of Interpreters or in number of memory modules.

(3) Device Controls (DC)

The DC resolves conflicts between Interpreters trying to lock to a device and checks the lock status of any Interpreter attempting a device operation. This unit handles up to four Interpreters and up to eight devices. System expansion using this module may be in number of Interpreters or in number of devices.

(4) Output Switch Network (OSN)

The OSN sends data and control from Interpreters to addressed memory modules (i.e., the OSN is a "demultiplexer"). This unit handles wires for up to four Interpreters and eight devices or memory modules. The number of wires for an OSN (and for an ISN) depends upon both the packaging and the "serialization" selected for the application.

(5) Input Switch Network (ISN)

The ISN returns data from addressed devices or memory modules to the Interpreters (i.e., the ISN is a "multiplexer"). This unit also handles wires for up to four Interpreters and up to eight devices or memory modules.

Shift Register (SR)

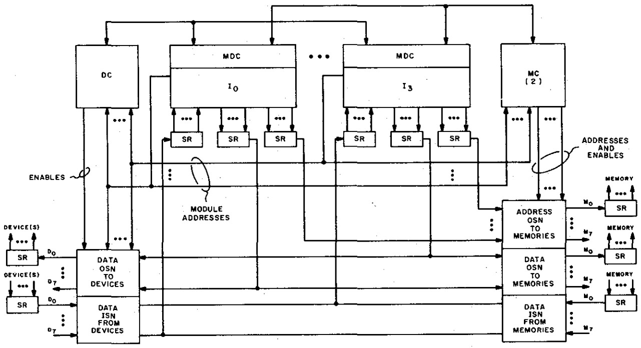
These units are optional and are parallel-to-serial shift registers or serial-to-parallel shift registers using a high frequency clock. These are used for serial transmission of data through the ISN's and OSN's. Their size, number and location are determined by system parameters.

Switch Interlock Block Diagram

Figure 6 is a block diagram of a Switch Interlock connecting up to four Interpreters to eight devices and eight memory modules. The shift registers shown are optional and may be eliminated with the resulting increase in the width of the ISN and OSN transfer paths. Although it is not indicated by this figure, the Switch Interlock is expandable in terms of number of Interpreters, devices, memory modules, and path widths.

Overall Switch Interlock Control

In an Interpreter based system utilizing one or more Interpreters, only Interpreters can issue control signals



NOTE: The widths of the ISN/OSN's are dependent upon the number of bits being transmitted serially.

Figure 6—Switch interlock

to access memories or devices. A memory module or device cannot initiate a path through the Switch Interlock. It may, however, provide a signal to the Interpreter via a display register or other similar external request device. Transfers between devices and memories must be via and under the control of an Interpreter.

Controls are routed from the Interpreters through the MDC to the MC and the DC which, in turn, check availability, determine priority and perform the other functions that are characteristic of the Switch Interlock. Data and addresses do not pass through the MDC.

Switch Interlock Timing

Events are initiated by the Interpreter for access to memories or devices. The Interpreter awaits return signals from the MDC. Upon receipt of these signals, it proceeds with its program. Lacking such positive return signals, it will either wait, or retry continuously, depending upon the Interpreter program (and not on the Switch Interlock). Any timeout waiting for a response may be performed by either the program or a device that will force a STEP in the micropogram after a preset length of time.

Among the significant signals which are meaningful responses to an Interpreter and testable as conditions are the following:

Switch Interlock has Accepted Information	The address and data output registers of the Interpreter may be reloaded and a memory or device has been connected.
Read Complete	Data are available to be gated into the input data register of the Interpreter.

The rationale for this "handshaking" approach is consistent with the overall Interpreter-based system design which permits the maximum latitude in the selection of memory and device speeds. Thus the microprogrammer has the ability (as well as the responsibility) to provide the timing constraints for any system configuration.

Device Operations

The philosophy of device operations is based upon an Interpreter using a device for a "long" period of time without interruption. This is accomplished by "locking" an Interpreter to a device. The ground-rules for device operations are listed below:

(1) An Interpreter must be locked to a device to which a read or a write is issued.
(2) An Interpreter may be locked to several devices at the same time.
(3) A device can only be locked to one Interpreter at a time.
(4) When an Interpreter is finished using a device, it should be unlocked so other Interpreters can use it. The exception is the case where devices locked to a failed Interpreter might be unlocked with a "privileged" instruction by another operative Interpreter.

Memory Operations

Memory modules normally cannot be locked and are assumed to require minimum access time and a short "hold" time by any single Interpreter. Conflicts in access to the same module are resolved in favor of the Interpreter that last accessed the module, or otherwise in favor of the highest priority requesting Interpreter. Once access is granted, it continues until that memory operation is complete. When one access is complete, the highest priority request is honored from those Interpreters then in contention. The Interpreter completing access is not able to compete again for one clock. Thus the two highest priority Interpreters are assured of access. Lower priority Interpreters may have their access rate significantly curtailed. This problem is resolved through careful allocation of data to memory modules.

Other Multi-Interpreter System Hardware

The executive concept described in the latter part of this paper requires special hardware features. Although in a multiple Interpreter system more than one Interpreter can execute executive programs at one time, certain tables used by the executive must be locked for modification by only one Interpreter at a time. The Interpreter remains in this "table modifying" state for a short time, thus minimizing executive conflict delays. The locking capability is implemented with two "global condition bits" per Interpreter which are chained to other Interpreters in a priority scheme. These global condition bits are 2 of the 16 testable condition bits in each Interpreter. Each bit can be set in only one Interpreter at a time and must be programmatically reset (the other condition bits in each Interpreter are reset

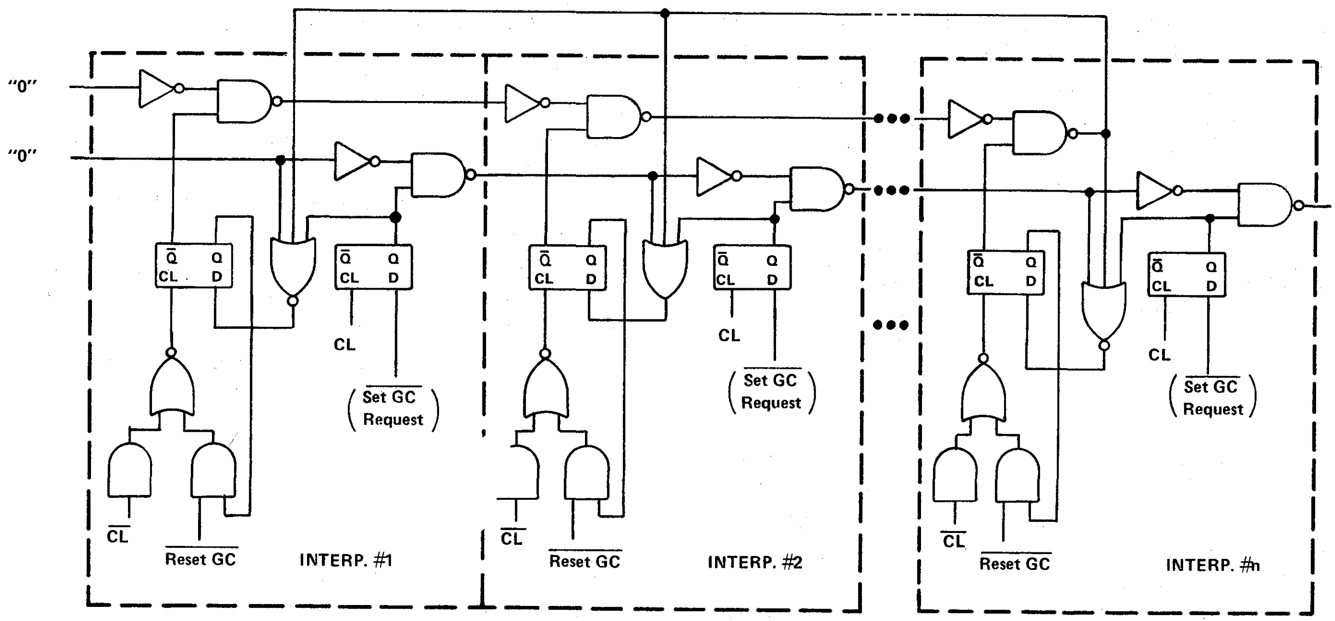


Figure 7—Global condition priority resolver

when they are tested). An Interpreter instruction containing the "Set Global Condition Bit" operation will set the specified global condition bit in that Interpreter only if that bit is not set in any Interpreter and no other higher (wired) priority Interpreter is requesting the same bit to be set in its own Interpreter.

Figure 7 shows the method of resolving priorities for each of the global condition bits. The instruction to set a global condition bit is actually a *request* to set a global condition bit. This request is latched for one clock time during which time a decision is made to honor this request or not. The global condition bits are programmatically reset independent of other Interpreters.

The top string of horizontally connected gates (Figure 7) OR's is the corresponding global condition bit from all Interpreters together. The other string of horizontally connected gates in the Interpreters is the wired global condition setting priority. This priority could be wired differently for the two global condition bits. It should be noted that in a system with many Interpreters, these two "carry" chains could have a delay long enough to affect the Interpreter clock rate. In these cases, a carry-lookahead scheme could be used to speed up this path.

It should also be noted that such a scheme could be implemented alternately via programming using Dijkstra's semaphores,[7] or using memories with a read-modify-write cycle to insure one processor wasn't reading a memory location in the interval between a read of that location by another processor and a subsequent write of a modified value.

One more of the testable condition bits in each Interpreter is wired to provide an additional inter-Interpreter signal. This bit is called the Interrupt Interpreters bit and is simultaneously set in all Interpreters by an operation originating from any Interpreter. This bit is reset in an Interpreter when tested in that Interpreter.

The global condition bits and the interrupt Interpreters bit are the only multiprocessing hardware features which are part of an Interpreter. Special purpose logic in an Interpreter is thus minimized, making it inexpensive and flexible enough to be used across a wide spectrum of architectures from simple stand-alone device controllers through multiprocessors. This versatility increases the quantity of Interpreters being produced, which in turn lowers the unit cost of an Interpreter.

Special devices (connected through the Switch Interlock) needed for multiprocessing are an interrupt display register and a real-time clock with time-out capability. The interrupt display register is needed due to the limited number of externally settable condition bits in each Interpreter. Such a display register is read as a device by any Interpreter. The response of the display register is a function of the interrupt handling philosophy of the particular application and the design of such a device could be varied without affecting the basic design of the Interpreter or the Switch Interlock.

The same philosophy is true of the real-time clock. Here, the intent of such a device is to provide an external counter for all Interpreters as well as a means of forcing a program STEP in an Interpreter if that Interpreter didn't periodically reset its real-time clock. This would prevent an Interpreter from waiting forever for a response from an inoperative memory or device.

MULTI-INTERPRETER CONTROL PROGRAM

The Multi-Interpreter Control Program is a simple, yet comprehensive, operating system characterized by the following capabilities:

(1) Multiprogramming and multiprocessing
(2) Fully automatic operation with manual intervention capability.
(3) Error recovery with no loss of data.

Multiprogramming and multiprocessing have characterized Burroughs operating systems for many years. In previous systems, input/output functions and data processing functions have been performed in physically different hardware modules; I/O modules for the former and processor modules for the latter. In the Multi-Interpreter System, however, I/O control and processing functions are all performed by identical Interpreters, and any Interpreter can perform any function simply by a reloading of its microprogram memory. In the Multi-Interpreter Control Program I/O operations simply become tasks which are indistinguishable to the control program from data processing tasks except that they require the possession of one or two I/O devices before they can begin to run. (A task is defined as an independent microprogram and its associated "S" level program and data, which performs explicit functions for the solution of user problems.) Whenever an Interpreter is available it queries the scheduling tables for the highest priority ready-to-run task, which may be an I/O task, a processing task, or a task which combines both processing and I/O functions.

The control program operates automatically, as well as under the control of the operator. The operator may enter new tasks, delete old ones, or change the priority of any task. He may add or delete hardware modules, and call for diagnostic programs to be run. The operator enters his commands through either the card reader or the alphanumeric display console.

The Multi-Interpreter Control Program includes an automatic error detection and recovery capability. All data is stored redundantly to avoid loss of data should a failure occur. The control program maintains this redundancy, in such a way that a restart point is maintained at all times for every task.

Figure 8 defines the flow of tasks through an Interpreter.

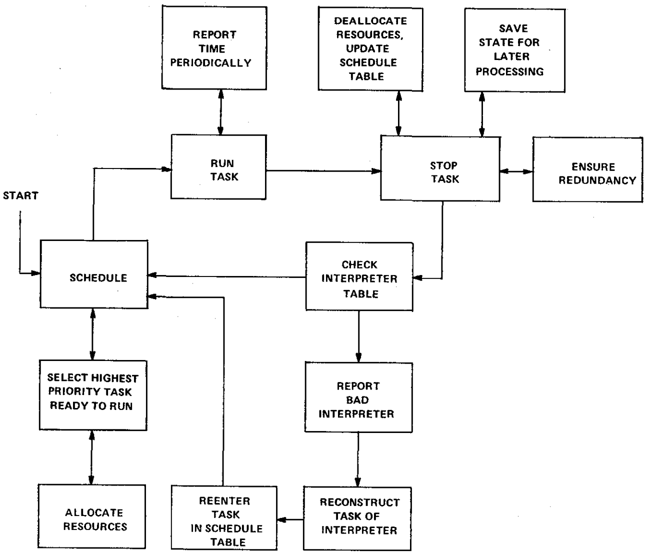

REPORT TIME PERIODICALLY
DEALLOCATE RESOURCES, UPDATE SCHEDULE TABLE
SAVE STATE FOR LATER PROCESSING
RUN TASK
STOP TASK
ENSURE REDUNDANCY
START
SCHEDULE
CHECK INTERPRETER TABLE
SELECT HIGHEST PRIORITY TASK READY TO RUN
REPORT BAD INTERPRETER
ALLOCATE RESOURCES
REENTER TASK IN SCHEDULE TABLE
RECONSTRUCT TASK OF INTERPRETER

Figure 8—Flow of tasks through an interpreter

Task scheduling and initiation

Every time an Interpreter has no task to perform it scans a Task Table and locates the highest priority ready-to-run task. If there is no ready-to-run task this Interpreter runs the diagnostic program, then (if the diagnostic program indicates no malfunction) again attempts to schedule a task. When the Interpreter finds a task which can be run the appropriate table entries are updated. The Interpreter then loads its microprogram memory with the task's code and begins to execute this code. The task code must periodically cause the Interpreter to "report in" to a central "timeout" table.

Task termination

The task retains the Interpreter until the task is due to be suspended (assuming that a timeout does not occur). The task first stores its state, then causes the Interpreter to load its microprogram memory with a portion of the control program. This recopies changed memory areas to ensure the redundancy required for error recovery, deallocates resources which the suspended task had used, and updates system tables. Then the Interpreter looks for another task.

Supervisory control

The operator can add or delete tasks or resources as well as enter the data and code which individual tasks require. Commands and data may be entered via either the card reader or an alphanumeric display. Each of these devices is "owned" by a system task which inputs data for the control program and for the other tasks in the system. Each entry via the card reader begins with a control card which specifies the nature of the entry. When the entry is data or program, the control card specifies the task and the area in that task where subsequent cards are to be loaded.

Commands and data are entered from the alphanumeric display in essentially the same manner as from the card reader, a line on the display corresponding to a card.

Error recovery

The hardware of the Multi-Interpreter System detects failures in memory modules and in I/O devices. When an I/O device fails, it is indicated as "not available" and a message to this effect is sent to the operator (via both printer and display). The device will remain "not available" until the operator enters a message to the contrary. The task whose running caused the malfunction to be detected is immediately aborted, but it will again run when the needed I/O device or an alternate is available. The redundant copies of storage areas serve as the restart point for this rerun. (The primary copies cannot be used as the task was aborted at other than a normal suspension point, hence not at a valid restart point.)

When the core memory module fails while a task is using it, the Interpreter which was running this task marks the failed module "not available", and causes the initiation of an error message to the operator. The task which was being run is aborted. This task will be reinitiated from the redundant core memory areas. Interpreter failures are detected in either of two ways: by the diagnostic program or by a timeout method, in which every Interpreter "checks in" to a central table at periodic intervals, and every Interpreter checks that every other Interpreter is checking in on schedule. When an Interpreter fails it halts (or is halted), and the task (if any) which it had been running is aborted, and will eventually be reinitiated at its restart point by an operative Interpreter.

The system can thus recover from a failure with no loss of data, provided the primary and alternate storage areas of a task are not both lost.

Tables

The operation of the Multi-Interpreter Control Program can be seen in more detail by examining the tables which it utilizes. These tables are divided into two classifications: system (global) tables, and task tables. The former are located in core memory in a segment called the "System Control Segment", and the latter for each task are located in a "Task Control Segment" for that task.

For error-recovery purposes all of these tables are stored redundantly in two different memory modules. Therefore there are actually two "System Control Segments", a primary one and an alternate one. The two copies are identical at all times and whenever one is updated the other is also. In a system that includes bulk storage this redundancy would be maintained in the bulk storage, and only one copy of any segment would be in core memory when the task is running.

The tables for each task are similarly stored redundantly so that there are two copies of each Task Control Segment. Included in each Task Control Segment are pointers to all of the data and program areas which this task may use. These areas may comprise one segment or many contiguous segments, and may be used either solely by this task or shared with other tasks. Each such area is also stored redundantly.

Figure 9 illustrates how the core memory is utilized, and shows the System Control Segment and its relationship to the Task Control Segments. (The term "segment" refers to a 256 word area of core memory.) The System Control Segment includes a pointer to each Task Control Segment, which in turn includes the pointers to each area used by the task.

Memory modules are presently paired, where both modules of the pair are allocated identically, so that the primary Task Control Segment and the redundant Task Control Segment for the same task have the same segment number. The same applies to data and program areas.

System control segment

The System Control Segment is illustrated in more detail in Figure 10. This segment is always located in segment zero (the first segment) of some memory module. Its redundant copy is always located in segment zero of some other module. Segment zero in all modules is reserved for possible use as the System Control Segment or its alternate copy. If the module containing the prime or redundant copy of the control segment should fail, any remaining module is selected and its segment zero is overwritten with the contents of the System Control Segment so that redundancy is immediately restored. Word zero in segment zero of every memory module contains pointers to both the prime and al-

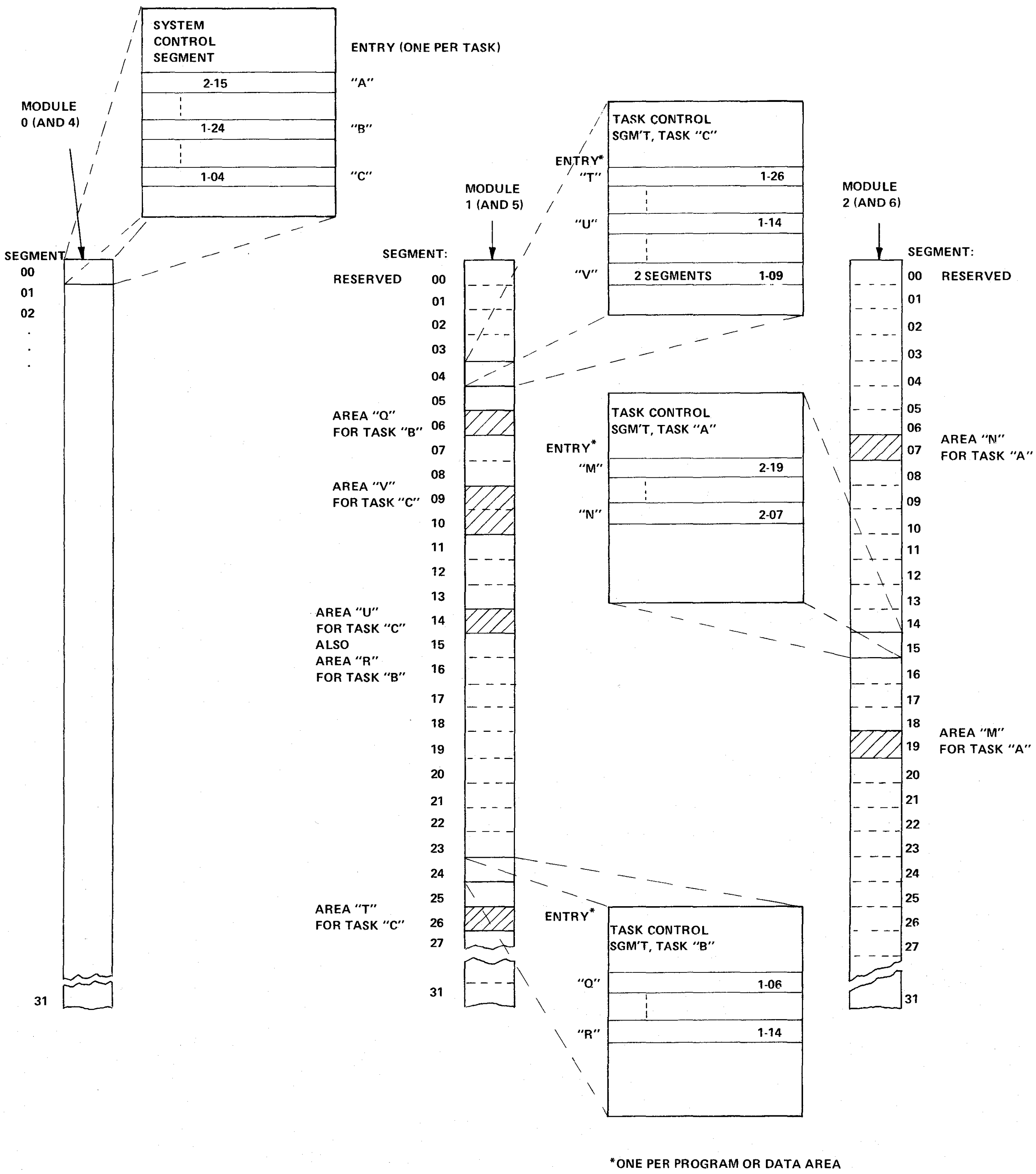


Figure 9—Memory utilization

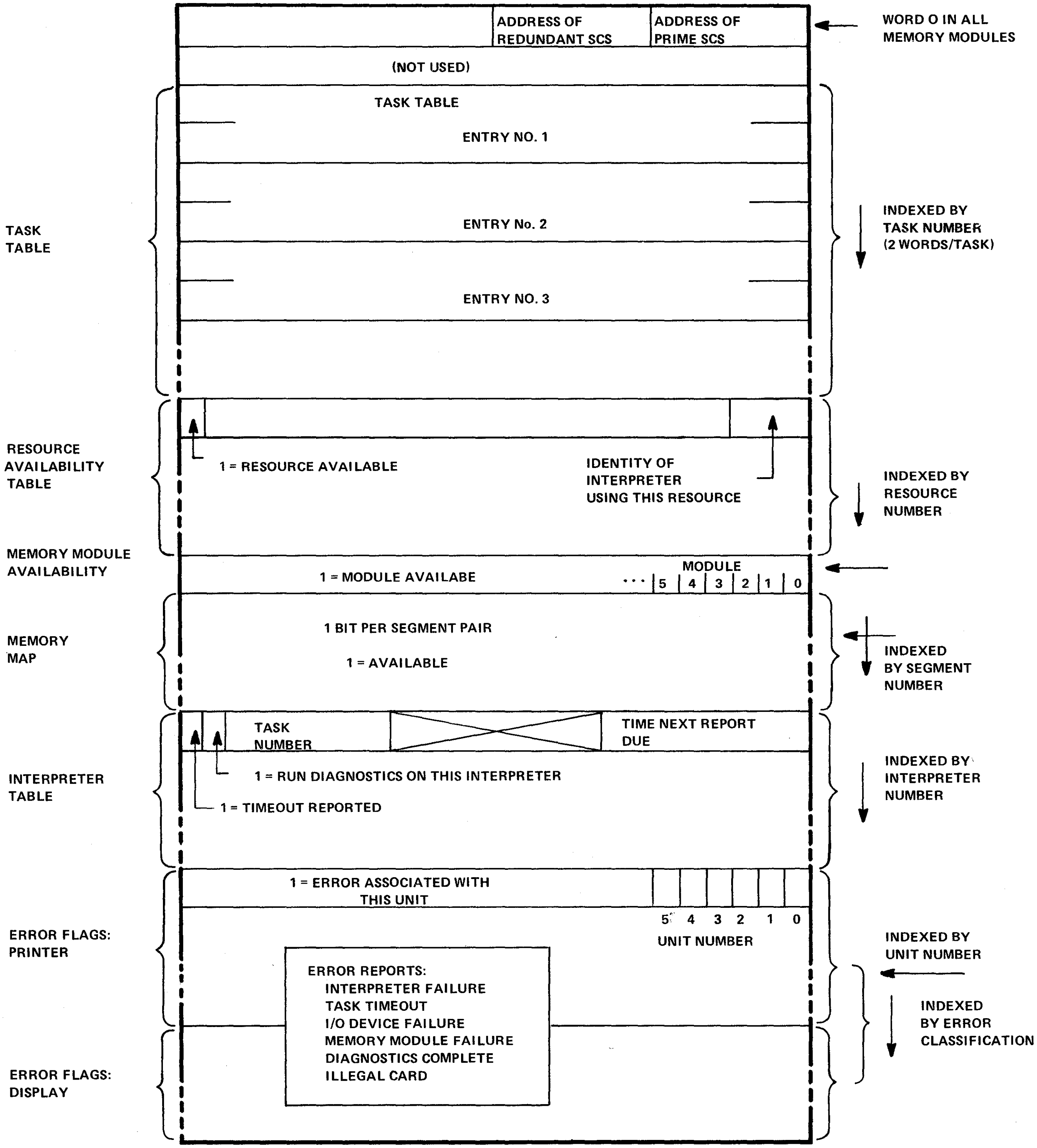


Figure 10—System control segment

ternate System Control Segments, so that any Interpreter can "find its way back" to the System Control Segment should it become "lost" because of a memory module failure. Word zero of the System Control Segment itself is no exception, and includes these pointers.

Resource Availability Table

This table contains one entry for every hardware resource. The entry includes a flag-bit, which indicates whether this resource is available or in use, and a field which, when the resource is in use, gives the identity of the Interpreter which is using it.

Memory Module Availability Table

This one-word table consists of one bit for every memory module in the system. This bit indicates whether or not the module is "up" (available for use).

Memory Map Table

This table contains one bit for every usable segment in core memory. A "1" in the bit position corresponding to a particular segment means that the segment is unassigned. Since modules are paired, a single bit in this table actually represents two segments, the prime segment and the segment in the paired module which holds the redundant copy.

Interpreter Table

This table has one entry for every Interpreter, and is used primarily to implement the "reporting" scheme used to detect Interpreter malfunctions. Every task run on the system is written so that, if it must run for "N" or more seconds, the Interpreter will "report in" to its entry in the Interpreter Table at intervals of less than "N" seconds. This reporting consists of replacing the "Time Next Report Due" field with the present time plus "N" seconds. Every Interpreter, when it completes its present task and is ready to find a new one, scans the Interpreter Table (after first updating its own entry in this table), looking for overdue entries. If any are found, they are reported to the supervisor. In addition, the most significant bit in the entry which was found overdue is set, so that other Interpreters will not make this same report.

Each entry in the Interpreter Table also includes two other fields. The first of these is a 1-bit field indicating that diagnostics are to be run on this Interpreter. This bit becomes set from a command entered by the supervisor. When set, the Interpreter will run diagnostics at the completion of the present task. Once diagnostics have been run the Interpreter causes a message to be sent to the supervisor indicating the test results, and then resets the "Run Diagnostics" bit.

The final field in an Interpreter Table entry gives the identity of the task which that Interpreter is currently running.

Error Flags Table

This table indicates the error messages which are to be sent to the operator. Such messages are sent both via the line printer and via an alphanumeric CRT display which is only used by the operator. Each portion of the table consists of one word for every type of error. Within each such word, the individual bits form an indexed vector which indicates more specific error information, normally the unit number of the failed unit. When any type of error is detected, the appropriate bit is set in both the display and printer portions of the table. The actual printout is accomplished by a specific task, the Error-Print task. This task (tagged "to be run") runs as soon as the printer and an Interpreter are available. When this task runs it scans the entire printer portion of the Error Flags Table, and for each bit encountered formats and prints the appropriate message and then resets the bit. This process continues until all of the bits are reset. There is also an Error Display task which functions in this same way to display error messages on the CRT screen.

Changing the System Control Segment

Whenever any change to a table in the System Control Segment must be made, the segment is locked. (This locking is implemented using the "global condition" bits found in every Multi-Interpreter System.) The required writing then takes place, to both the prime and the redundant copy of the table, and the segment is then unlocked.

The task table

The Task Table contains information about every task in the system. Figure 11 indicates the contents of an entry in this table, which consists of two words. The first word contains the information which an Interpreter in the process of finding a new task must examine to determine whether this task is the one which should be

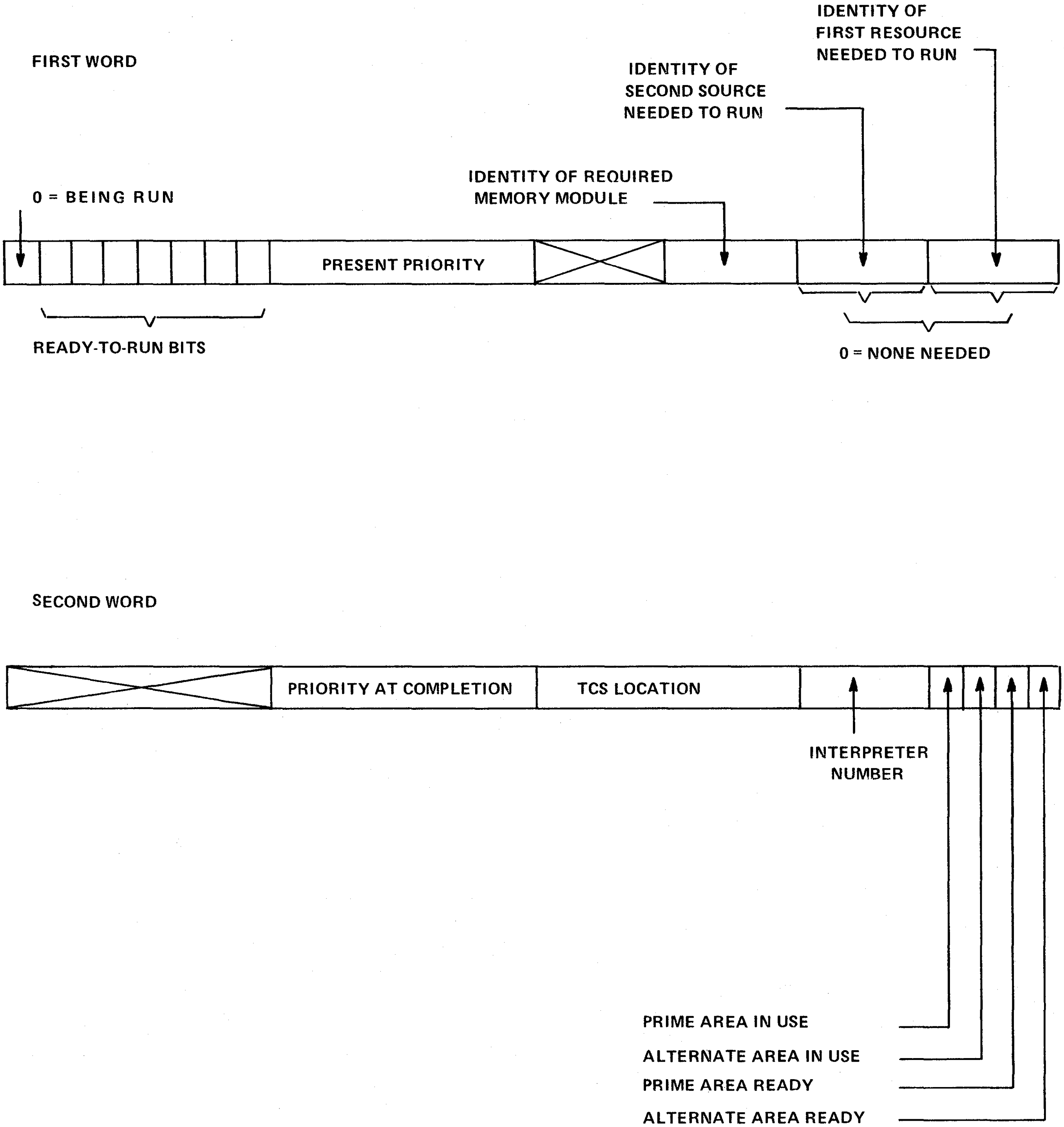


Figure 11—Task table entry

selected. The most significant bit in this first word is the "not-running" bit. This bit is reset whenever the task is running and set at all other times. Next are the "ready-to-run" bits. These bits must all be ones for the task to be scheduled. If one or more of its bits are zero, the task lacks something, other than an I/O device, which it requires for running. For example, if the task requires data to be supplied by some other task, one of its ready-to-run bits would remain reset until the completion of the other task. Similarly, if the task in question is awaiting data from the card reader, a bit will be left reset and will be set by the Card Reader Task after the data has been loaded. These bits have no system-wide assignments, but are assigned for each application. All unassigned bits remain set at all times. Any task can reset any of its own bits, and can, upon its suspension, set bits for other tasks.

The next field in the Task Table Entry is the "present-priority" field. Each task has a priority, and that ready-to-run task with the highest priority is the one which is selected.

In order for a task to be "ready-to-run", not only must all of its ready-to-run bits be set, but also the core memory which it requires, and any I/O devices which it may need must be available. The remaining fields of the first Task Table Entry word deal with these items. There is a field which specifies the memory which this task requires, and two fields which can specify up to two I/O devices which it also requires. Before the task can be considered "ready-to-run", the required areas of memory, and all required I/O devices must be available.

The second word of the Task Table Entry contains additional information about the task. The first such field specifies the priority to which the task reverts after it has completed running. (A priority-at-completion of zero indicates that the task is to be dropped from the system at completion.) Next is the location of the Task Control Segment for this task, as discussed in connection with Figure 9. Another field contains the identity of the Interpreter which is running this task.

The final field in the Task Table Entry gives the status of the core memory areas used by this task. The

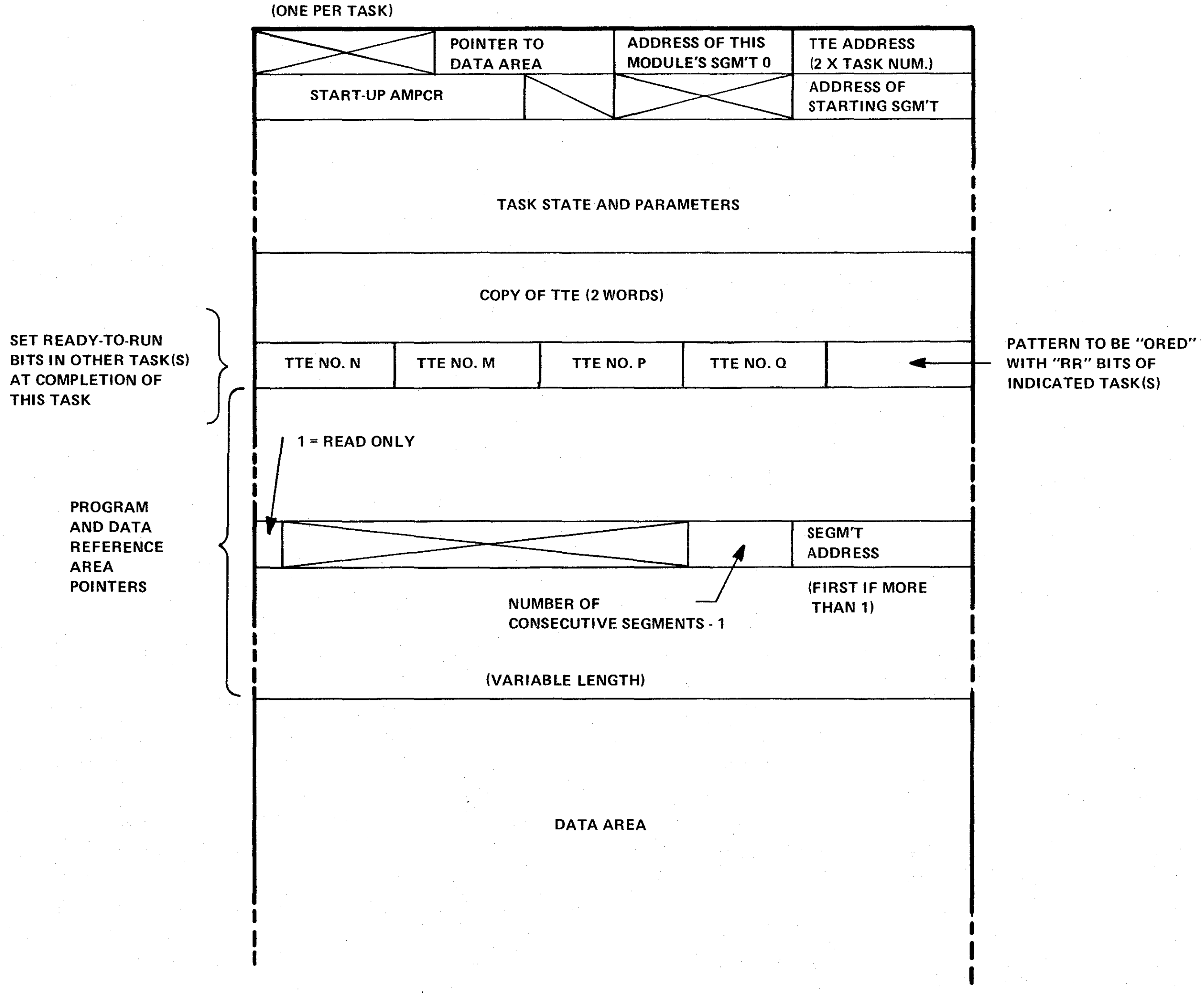

(ONE PER TASK)
POINTER TO DATA AREA
ADDRESS OF THIS MODULE'S SGM'T 0
TTE ADDRESS (2 X TASK NUM.)
START-UP AMPCR
ADDRESS OF STARTING SGM'T
TASK STATE AND PARAMETERS
COPY OF TTE (2 WORDS)
SET READY-TO-RUN BITS IN OTHER TASK(S) AT COMPLETION OF THIS TASK
TTE NO. N
TTE NO. M
TTE NO. P
TTE NO. Q
PATTERN TO BE "ORED" WITH "RR" BITS OF INDICATED TASK(S)
1 = READ ONLY
PROGRAM AND DATA REFERENCE AREA POINTERS
SEGM'T ADDRESS
(FIRST IF MORE THAN 1)
NUMBER OF CONSECUTIVE SEGMENTS - 1
(VARIABLE LENGTH)
DATA AREA

Figure 12—Task control segment

Task Control Segment and all program and data areas associated with task exist in two different memory modules. When the task runs, only one module is actually used, and the other holds the state of this task as it existed after it was last processed. This then serves as a "restart point" should the present processing be prematurely terminated by a failure. This alternate area is not changed until this present processing has been successfully terminated and the primary copy of the task's Task Control Segment updated to form a consistent whole which can itself be used as a restart point. Only then is the Task Control Segment and all data areas copied from the primary to the alternate module.

Task control segment

Figure 12 indicates the contents of the Task Control Segment. The first field points to the end of the Reference Area, since the size of the Reference Area may differ from task to task. (The otherwise unused end of the segment may be used as a work area by the task.) The second field gives the address of this module, so that, when the task is completed, word zero of segment zero can be found. (This word must be accessed in order to locate the System Control Segment.) The third field in this first word gives the identity of the present task, and enables the Task Table Entry for this task to be located once the System Control Segment has been found.

The next section of the Task Control Segment holds the state of the task, and also task parameters. First is the address of the task code at which the task is to be restarted. This task code then interprets the remainder of the state information, if any.

The next section of the Task Control Segment contains a copy of the Task Table Entry (from the System Control Segment), put there by the control program when the task is initiated. The task may then conveniently modify its Task Table Entry. For example, the task may reset one or more of its "ready-to-run" bits, may indicate the identity of an I/O device which is required for further processing, or may even change its priority. When the task is suspended, and it runs until it suspends itself (unless it is suspended because of a time-out), the Task Table Entry in the System Control Segment will be updated by the control program to correspond to the Task Table Entry as found in the Task Control Segment.

In addition to modifying its own Task Table Entry, the task may, upon its suspension, cause Task Table Entries for other tasks to have one or more ready-to-run bits set. This is accomplished through the next word in the Task Control Area. This word allows the task to specify up to four other tasks, and to supply a bit pattern to be "ored" with the ready-to-run bits for each of these tasks. In this way the completion of one task can be used to initiate another.

The final section in the Task Control Segment is the reference area, which points to data and program segments which this task may use. Each reference area, in addition to specifying the address of the segment(s) to which it refers, also contains a "read-only" bit, and the control program does not update the alternate copy of any entry so tagged.

In summary, the Multi-Interpreter Control Program allows any number of Interpreters to operate concurrently in a multiprocessing and multiprogramming mode, allows tasks and hardware modules to be added and deleted by the operator, and provides error recovery capability.

A BUILDING BLOCK APPROACH TO SOFTWARE

There are two approaches that may be followed when developing an operating system for a multiprocessor with microprogrammable Interpreters. One is an approach taken in the paper by Davis and Zucker[1] as well as B. J. Huberman[8] where a machine language is developed. This language includes instructions which aid in the development of an operating system. Since operating systems are organized about tables, lists, queues and stacks the machine language developed allows for easy handling of tables and other data structures.

We have presently chosen to take a building block approach. The Multi-Interpreter Control Program which is our current system, presents the operating system as a set of modules which may be obtained and run by an Interpreter when needed. A new technique currently under development generates an operating system which is basically a set of independent "S" instructions which may be used by any task on a defined configuration of firmware and hardware.

The building block approach used in the hardware implementation of the multiprocessor is also being followed for the software development. A simple flexible method of system configuration is defined. This system has the ability to automatically select the appropriate units, either system microcode or system tables, from a prestored library of such units. This technique permits using only those units necessary to perform all the desired system functions for a particular set of tasks. System functions are requested as needed from a Parts List. The appropriate units are selected from the list to form a customized system from a set of standard building blocks.

A primary design criteria of the operating system (Manager) are flexibility, simplicity and reliability. The operating system must be flexible enough for all classes of problems to fit within its structure. All tasks must be able to access all facilities provided by the system with a given configuration of the hardware. Simplicity is necessary to allow the system to be maintained, changed, tested and initially coded with relative ease. The system architecture must be sufficiently straightforward so that the users can easily comprehend its structure and operation. The manager must also be reliable and the recoding of a single unit should not affect the rest of the system.

To achieve the above criteria a modular, distributed system manager has been defined. Each unit is a separate group of microinstructions designed to run in a multiprogrammed mix and must be validated as a standalone program. Extreme attention must be paid to modularity, allowing new units to be plugged in without causing bugs in the rest of the system. Just as the multiprocessor's hardware flexibility lies in its easily changeable hardware configuration so should the multiprocessor's software flexibility lie in its easily changeable software configuration.

The system is composed of a single structure, and a set of conventions for using this structure. Although systems will differ, both in type of units and hardware, the set of conventions and the general structure will always be used in the same way. The specific units needed for the manager functions will be plugged into the structure as needed.

The three segments defining the operating system are the Locator, the Parts List and the units. The units are the function modules of the system which do the desired operations for the users. This collection of modules is both programs and data (tables) normally thought of as

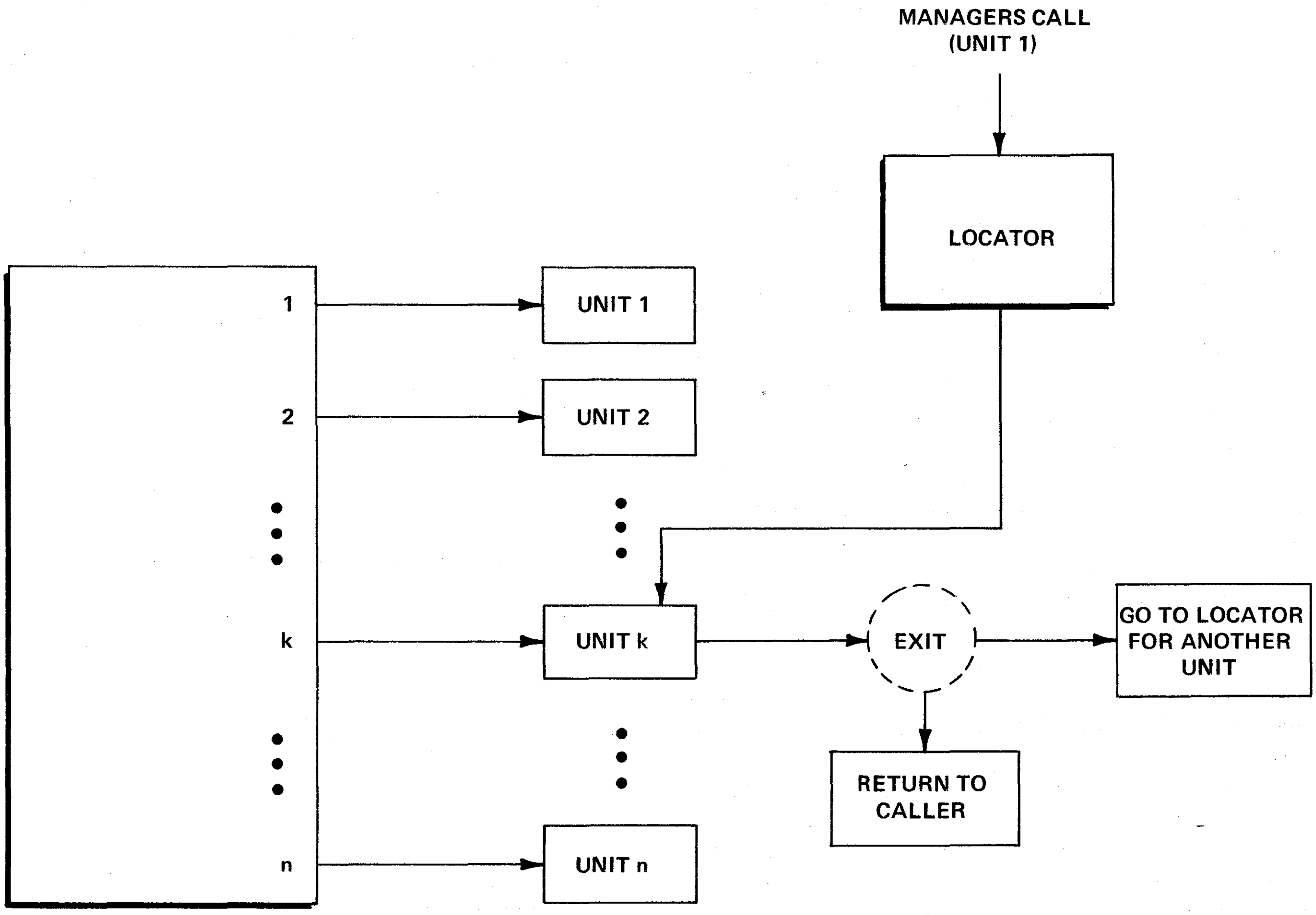


Figure 13—System manager

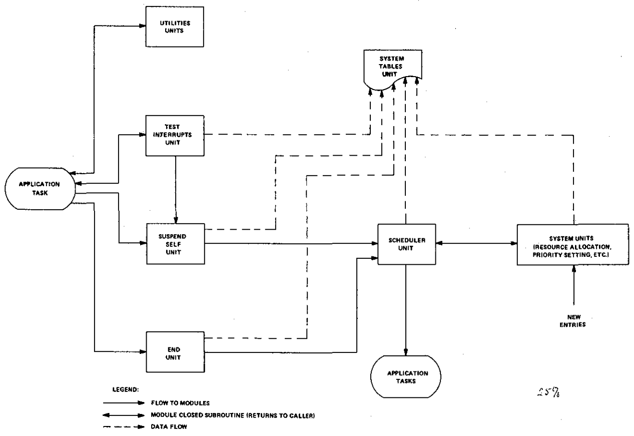


Figure 14—Program traffic in a processor

the operating system. The tasks may select a unit through the Locator. The Locator is a part of all tasks and can access all units with a parameter called the unit number which locates the required unit using the Parts List. The Parts List is a table containing the location of all units associated with a system. Each time a new system is required, a new Parts List and a new set of units are developed. However, the Locator remains the same for all systems independent of the kind of hardware or software in use (Figure 13).

The traffic flow within a single processor is shown in Figure 14. Each task will require the use of the system units to perform some known utilities for them. This includes functions like memory allocation, I/O formatting, conversions, editing, debugging, resource allocation, channel checking, etc. The utility units perform a function and then return to the caller.

Interrupts are soft in a Multi-Interpreter System. Testing for interrupts is processed by a standard interrupt testing unit called by a task. The detection of an interrupt may suspend the caller task or it may call another unit to process the interrupt before returning to the caller. User tasks may suspend themselves or be completed by calling the Suspend Self or End units. When a program has been suspended or ended the next highest priority program ready to run is selected by the scheduler unit.

When an Interpreter task is assembled, a Source Table is developed for all of the manager hardware dependent subroutines it may use, as well as all of the functions needed by its Interpreter. This table becomes input information to the operating system Librarian (Figure 15). Each microprogrammed task has such a table developed before it can run. When the task must access the library units, it indexes into the Source Table. The Source Table defines the library unit needed. Source Tables for all tasks are used as inputs to the Operating System Librarian. This information is used with a catalog of units to develop the Parts List for the system.

Each Source Table is modified to become a list of pointers to the Parts List (Figure 16) and to the allocation portion of the Locator in microprogram memory. As each unit is referenced during a task run, the Source Table addresses the allocation part of the Locator. Space is allocated in microprogram memory. The address pointing to the allocation routine is changed to point to the desired unit now transferred into microprogram memory. The unit can now be executed. Unless this unit is deallocated from microprogram memory, the Source Table now can point directly to the unit address in this memory.

To locate a unit for allocation in microprogram memory, the allocator uses the pointer in the Source Table to locate the pointer in the Parts List which points to the unit. When the unit is present in main memory it need only be copied into the required location in microprogram memory. When the unit is not present it must

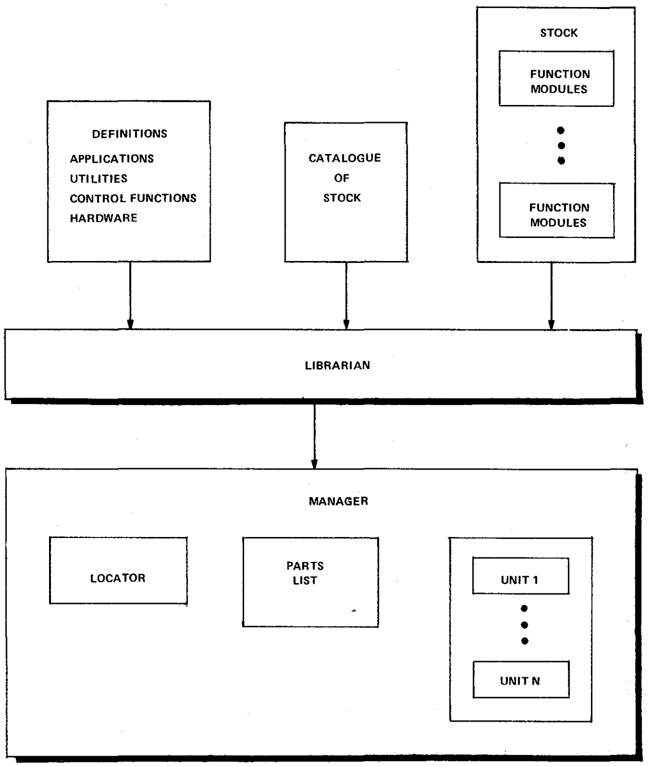


Figure 15—Automatic development of manager

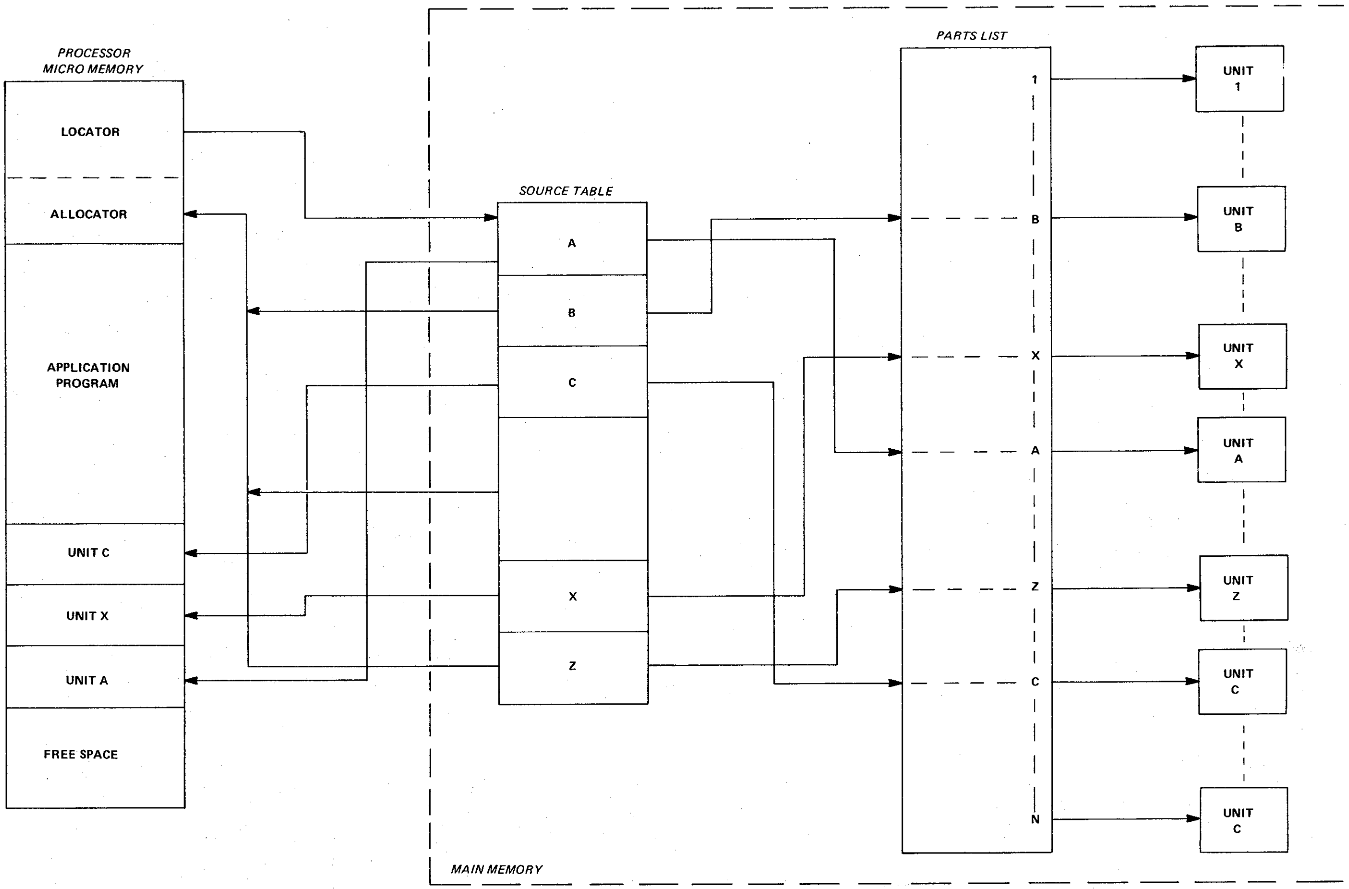


Figure 16—Single local manager

be read into memory from a peripheral device as defined in the Parts List. This device will be defined by the file directory (Figure 17) so that units may be accessed through a hierarchy of storage in a recursive manner.

Since all units of code in the Parts List are reentrant, all processors may have the same code at the same time without interfering with one another (provided they use different work areas). The work area is unique for each task and within each work area is a unique Source Table. Figure 18 shows the distributed local operating system with many Interpreters. Each Interprecer is: executing a different task, working in a unique work area, using a unique Source Table. All Interpreters share the same Parts List and have identical Locator sections. Any one unit can be unique to a task or shared by many tasks.

Since all system tasks are independent of each other, the parts that bind the system together are the system tables. These contain the important interfaces of the system and the means of communication between Interpreters and/or processes. The system tables define the three main attributes of the system: the tasks to run, the resources available, and the Interpreters running.

A Task Table is used for scheduling and termination of tasks at the global level. All information pertaining to the state, selection criteria, needs, and scheduling of a task will be found in this table. Intertask communication and task Interpreter communication is performed in the Task Table.

The Resource Table provides the information for allocating system resources among the different tasks by the various units.

An Interpreter Table contains information pertaining to each individual Interpreter. The Interpreters check on each other for the detection of malfunctioning hardware, as well as inter-Interpreter communication via this Table. An Interpreter may also use its table entry as temporary storage or for storing information about future tasks (e.g., an Interpreter may have to run routine diagnostics at the completion of its present task).

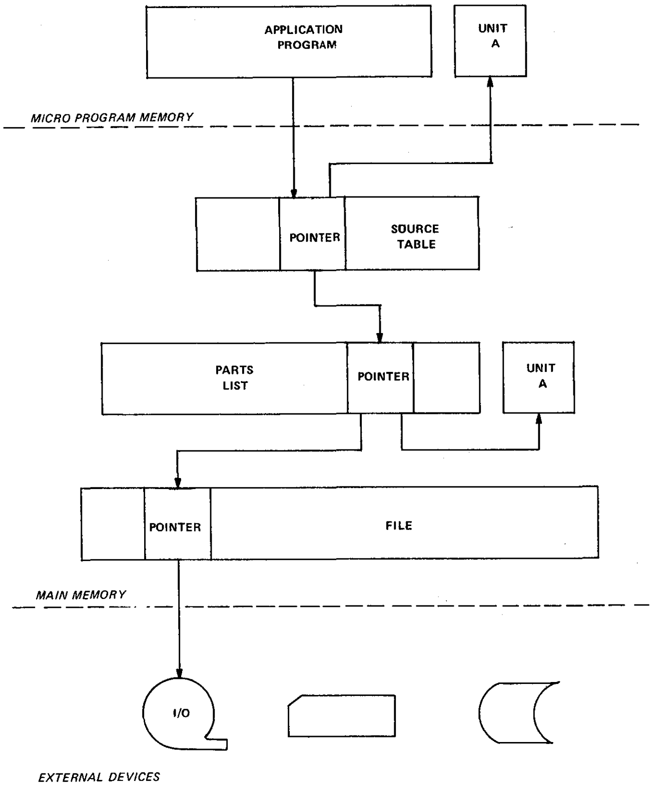


Figure 17—Accessing a unit

The building block approach assembles an operating system from a series of directly executable "S" instructions, called units, which can be executed by any task. These units may be different for different systems. A system which would want an I/O module could assign a high priority task the role of doing this job. Then all I/O system units would become descriptor building units or communication units instead of direct execution I/O units. In some systems, tasks would do their own I/O by having direct execution Parts List units while other tasks in the same system might assume an I/O module and pass descriptors to an I/O Task using other units.

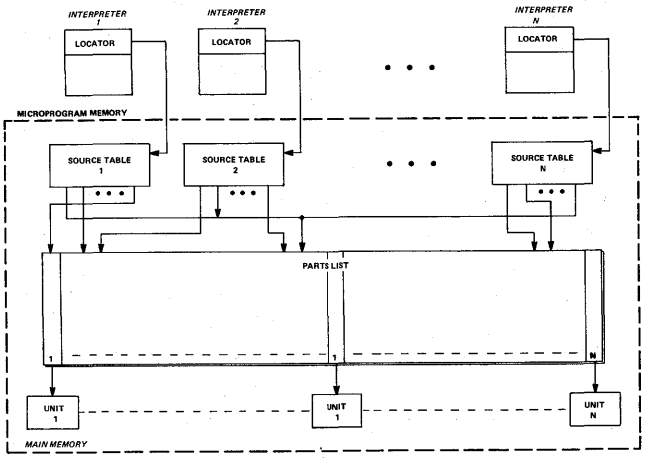


Figure 18—Distributed local managers system

Those tasks of an operating system which stand alone (e.g., I/O processor, system loaders, external communicators, etc.) are called and entered in the task table as if they were an application task with a high priority. Their ready to run bits can be set and reset by the system units.

The structure of the building block type operating system is a systematic way of constructing a manager so that its functions can be applied with identical units at all levels in a hierarchy of operating systems (e.g., course scheduling can be done at a global level while finer scheduling can be used at the next level). An Interpreter may use a unit to select a system for emulation, the task being that system's Operating System. Using the resources assigned to it at the global level, this Interpreter may choose to run its own set of tasks (which look like data to the global system). The Operating System of an emulation may be totally in the "S" language being interpreted or it may be running global units from the Parts List on its own resources assigned to it at the global level.

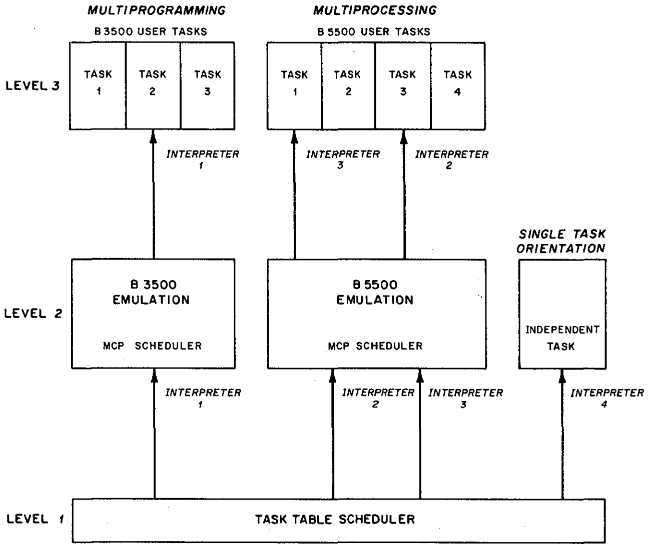


Figure 19—Interpreter control level

Figure 19 shows the control levels an Interpreter goes through to execute a user task. Interpreter 1 has been given resources and is scheduled at level 1 to run a B 3500 emulation. The initial task to be run is the operating system for the B 3500 called the MCP. It in turn receives data about the tasks to be run on a B 3500. The MCP sets up the tasks and executes them in a multiprogramming environment. Figure 19 shows task 2 as running. Interpreter 1 is now running a B 3500 program and is a B 3500 system which is totally independent of the rest of the system except for reporting status or requesting more resources.

Interpreters 2 and 3 have both been scheduled to become a B 5500 system. These Interpreters now have become a multiprocessing B 5500 system running the MCP and the B 5500 tasks. Interpreter 3 is running task 1 and Interpreter 2 is running task 3 of the B 5500 system's tasks in the B 5500 schedule.

Interpreter 4 has selected a task oriented emulation. This is a microprogram specifically oriented to doing a specific job. It needs no operating system but uses the units available to it in the Parts List for executing common functions or manager type functions.

A task must decide how critical its need is for quick access to a system unit. It may decide a unit is important and once it is located and stored in microprogram memory it is left there for the duration of the task. A task which is required to process real time interrupts may want the interrupt testing unit to be a part of its code and not have to indirectly address the unit via its Source Table. Such a task may directly assemble the interrupt unit from the library into its microprogram memory. Other units which are less critical will be brought from main memory each time they are accessed.

CONCLUSIONS

Microprogrammed systems in the past have been relatively simple and primarily suitable for small, dedicated tasks. A technique has been needed to interconnect many small microprogrammed processors into one system, and to control this array of processors so that it can dynamically and efficiently share a large load. This paper has presented such a technique. The Switch Interlock allows an array of Interpreters to be integrated into a unified system with many memories and peripheral devices. The type of software described here provides a means for controlling this unified system, and allows control programs to be "custom-tailored" to each application, providing maximum efficiency. Thus the flexibility of microprogramming can now be applied to large scale systems, and virtually any size system may be constructed with a smooth evolution from a small system to a large one. A degree of reliability and of simplicity in logistics and maintenance not previously possible in medium or large scale systems is provided due to repeated use of a small number of relatively simple module types.

Microprogrammed multiprocessing systems thus appear well suited for a wide variety of data processing applications.

REFERENCES

1 R L DAVIS S ZUCKER
Structure of a multiprocessor using microgrammable building blocks
Proc of National Aerospace Electronics Conference Dayton Ohio pp 186-200 May 1971

2 E W REIGEL D A FISHER U FABER
The interpreter—A microprogrammable processor
AFIPS Conf Proc (SJCC) vol 40 Montvale NJ AFIPS Press 1972

3 W A CURTIN
Multiple computer systems
Advances in Computers vol 4 F L Alt and M Rubinoff Eds New York Academic Press 1963

4 R C LARKIN
A minicomputer multiprocessing system
Proc of Computer Designers Conference Anaheim Calif pp 231-235 Jan 1971

5 *Seek flex preliminary design study, volume 1: System design and rationale*
Hughes Aircraft Co Ground System Group Report FR71-16-430 July 23 1971

6 J D MENG
A serial input/output scheme for small computers
Computer Design vol 9 no 3 pp 71-75 March 1970

7 E W DIJSKTRA
The structure of 'THE' multiprogramming system
Communication of ACM Vol 11 N 5 pp 341-346 May 1968

8 B J HUBERMAN
Principles of operation of the venus microprogram
Mitre Technical Report 1843 May 1 1970

The interpreter—A microprogrammable building block system

by E. W. REIGEL, U. FABER, and D. A. FISHER

Burroughs Corporation
Paoli, Pa.

INTRODUCTION

What is microprogramming?

Digital computing systems have traditionally been described as being composed of the five basic units: input, output, memory, arithmetic/logic, and control (see Figure 1). Machine instructions and data communicated among these units (as indicated by the solid lines in the figure) are generally well-known and understood. The control signals (as indicated by dashed lines in the figure), are generally less well-known and understood except by the system designer. These control signals generated in the control unit determine the information flow and timing of the system.

Microprogramming is a term associated with the orderly and systematic approach to the design of the control unit. The functions of the control unit include:

1. Fetching the next machine instruction to be executed from memory
2. Decoding the machine instruction and providing each microstep control
3. Controlling the gating of data paths to perform the specified operation
4. Changing the machine state to allow fetching of the next instruction.

The conventional control unit is designed using flip-flops (e.g., registers and counters) and gating in a relatively irregular *ad hoc* manner. By contrast the control unit of a microprogrammable computer is implemented using well structured memory elements; thus providing a means for well organized and flexible control.

Microprogramming is therefore a technique for implementing the control function of a digital computing system as sequences of control signals that are organized on a word basis and stored in a memory unit.

It should be noted that if this memory is alterable, then microprogramming allows the modification of the system architecture as observed at the machine language level. Thus, the same hardware may be made to appear as a variety of system structures; thereby achieving optimum processing capability for each task to be performed. The ability to alter the microprogram memory is called dynamic microprogramming as compared to static microprogramming which uses read only memories.

As can be seen in the following brief historical review, the concept of microprogramming was not widely accepted except academically during the 1950s. The primary reason for this was its high cost of implementation, especially the cost of control memories. From the mid-1960s to the present there has been a definite trend toward microprogrammable processors and more recently to dynamic microprogramming. This effort has been inspired by rapid advances in technology, especially control memories.

Brief historical review of microprogramming

1951 Wilkes[1] objective was "to provide a systematic approach and an orderly approach to designing the control section of any computing system." He likened the execution of the individual steps within a machine instruction to the execution of the individual instructions in a program; hence the term microprogramming. This view is hardware design oriented.

Lincoln Lab (see Van der Poel[2]) with different emphasis used the term microprogramming to describe a system in which the individual bits in an instruction directly control certain gates in the processor. The objective here was to provide the programmer with a larger

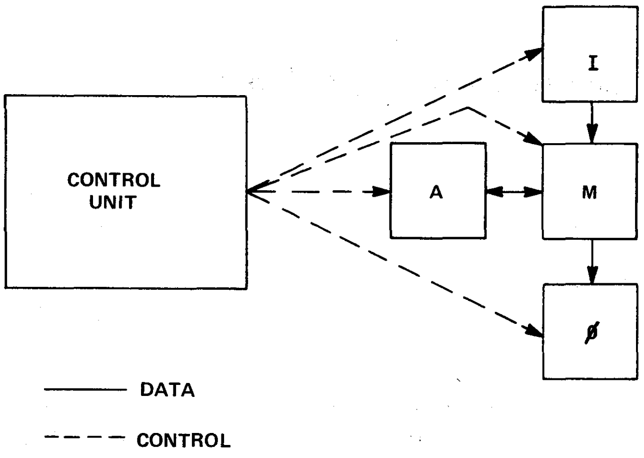


Figure 1—Five basic functional units of digital computing system

instruction repertoire. This view is software design oriented.

1956/7 Glantz[3] and Mercer[4] pointed out that through microprogram modifications the processor instruction set may be varied.

1958-1960 Blankenbaker,[5] Dinneen,[6] and Kampe[7] described simple computers based on Wilkes' model.

1961-1964 Great international interest was shown from U.S., U.K., Italy, Japan, Russia, Australia and France.

Feb. 1964 In Datamation[8–12] five articles appeared on microprogramming with emphasis on how it might extend the computing capacity of small machines.

1964 IBM System 360 (Stevens[13]) demonstrated that through microprogramming, computers of different power with compatible instruction sets could be provided (used read only storage).

1965 Melbourne and Pugmire[14] described microprogramming support for compiling and interpreting higher level programming languages.

1965 McGee and Peterson[15] pointed out the advantage of using an elementary microprogrammed computer as a peripheral controller, i.e., as an interface between computers and peripheral devices.

1965-1966 Green[16] and Tucker[17] described emulation of one machine on another through microprogramming.

1967 Opler[18] coined the term "firmware" for microprograms designed to support software and suggests the increased usage of microprogramming and describes its advantages.

1967 Hawryszkiewycz[19] discussed microprogram support through special instructions for problem oriented languages.

1967 Rose[20] described a microprogrammed graphical interface computer.

1968 Lawson[21] discussed program language oriented instruction streams.

1969 Wilkes[22] and Rosin[23] provided surveys of the microprogramming advances.

There were also announcements of many new microprogrammed computers (e.g., Standard Computer—Rakoczi[24]).

1970 Husson[25] provided the first textbook on microprogramming.

1971 Tucker and Flynn[26] pointed out advantages of adapting the machine to the task through microprogramming.

July 1971 The IEEE Transactions on Computers offered a special issue on microprogramming.

Interpreter design philosophy

The main objectives in the design of Burroughs' Interpreter were simplicity, versatility, technology independence, and modularity. The reasons for these objectives and their realization in the resulting machine are stated in Table I.

The basic concepts that characterize the Interpreter are its modular structure using simple building blocks and its application to a wide range of tasks through its dynamic microprogramming.

A system is composed of a number of Interpreters, main memories, input/output devices, and a data exchange called the switch interlock. The switch interlock allows each Interpreter to communicate with all memories and devices and consists of controllable data paths and conflict resolution logic for access to these paths. The switch interlock (described in a companion paper) is an extension of the designs that have been used successfully for many years in Burroughs' multiprocessing systems (e.g., B 5500 and D 825).

INTERPRETER HARDWARE BUILDING BLOCKS

The Interpreter is composed of three logic package types; namely, the **L**ogic **U**nit (LU), the **C**ontrol **U**nit (CU), and the **M**emory **C**ontrol **U**nit (MCU). The microprograms which provide the control functions are contained in two conventional memories; namely, the

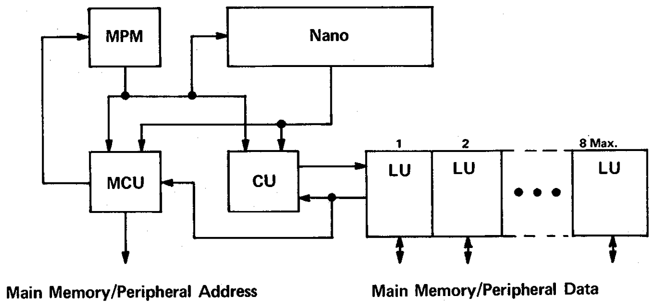
TABLE I

Desired Characteristic	Reasons for Objectives	Resulting Characteristics
Simplicity	Omission of special functions not applicable over a wide range of applications Ease of maintainability: Training Documentation Spare parts	Each Interpreter consists of: a. Three logic package types (each less than 1000 gates) Logic Unit (LU) – Control Unit (CU) – Memory Control Unit (MCU) – b. Standard memories for microprogram storage (e.g. Read/Write or Read Only)
Versatility	Increase volume of production resulting in reduction of cost Decrease design time and effort by eliminating the need for new hardware developments for each new product	Peripheral controller I/O controller Central processor Special function processor Emulator of existing systems Direct execution of languages
Technology Independence	To provide smooth transition to LSI when economically justified To meet variety of system cost/performance requirements with optimum circuitry	Partitioning allows implementation in variety of packaging schemes (e.g., SSI, MSI, LSI) Performance may be varied by simply implementing in different circuit family (e.g. MOS, TTL, ECL)
Modularity	System – to provide smooth and unlimited system growth Internal – to provide a variety of machine word sizes as needed by the application	System modularity provided by modular Switch Interlock Internal modularity provided by use of multiple logic units (LU).

Microprogram Memory (MPM) and the Nano program Memory (Nano or NM). These units and their interconnections are shown in Figure 2.

The unique split memory scheme for microprogram memories allows a significant reduction in the number of bits for the microinstruction storage. It should be noted, however, that a single microprogram memory scheme (MPM and Nano combined) may also be used potentially increasing the clock rate of the system. In addition, the cycle rates of the memories may also be altered, to gain speed or reduce cost, without any redesign of the logic packages. In fact, a variety of memory organizations (single memory and different split memory configurations) and memory speeds have been implemented thus providing a range of cost/speed trade-offs.

The LU performs the shifting and the arithmetic and logic functions required, as well as providing a set of scratch pad registers and the data interfaces to and from the Switch Interlock (SWI). Of primary importance is the modularity of the LU, providing expansion of the word length in 8-bit increments from 8 bits through 64 bits using the same functional unit. The CU contains a condition register, logic for testing the conditions, a shift amount register for controlling shift operations in the LU, and part of the control register used for storage of some of the control signals to be sent to the LU. The MCU provides addressing logic to the Switch Interlock for data accesses, controls for the selection of microinstructions, literal storage, and counter operation. This unit is also expandable when larger addressing capability is required.



Each Interpreter consists of:

- Three types of logic packages
  - ▲ Logic Unit (LU)
    - – Registers A1/A2/A3/B/MIR
  - ▲ Control Unit (CU)
    - – Registers SAR/Command/Condition
  - ▲ Memory Control Unit (MCU)
    - – Registers MPCR/AMPCR; BR1/BR2/MAR; LIT/CTR
- Two memory packages (standard available units — R/W or R/O)
  - ▲ Microprogram Memory (MPM)
  - ▲ Nanoinstruction Memory (Nano)

Figure 2—Interpreter building blocks

Logic unit (LU)

A functional block diagram emphasizing the LU is shown in Figure 3. Registers A1, A2, and A3 are functionally identical. They temporarily store data within the Interpreter and serve as a primary input to the adder. Selection gates permit the contents of any A registers to be used as one of the inputs to the adder. Any of the A registers can be loaded from the output of the barrel switch.

The B register is the primary external input interface (from the Switch Interlock). It also serves as the second input to the adder, and can collect certain side effects of arithmetic operations. The B register may be loaded with any of the following (one per instruction):

1. The barrel switch output
2. The adder output
3. The external data from the Switch Interlock (or control panel switches)
4. The MIR output (the MIR is the output data register to the Switch Interlock)
5. The carry complements of four- or eight-bit groups (with selected zeros for use in decimal arithmetic or character processing)
6. The barrel switch output ORed with two, three, or four above.

The output of the B register has true/complement selection gates which are controlled in three separate sections: the most significant bit, the least significant bit, and all the remaining central bits. Each of these parts is controlled independently, and may be either all ZEROs, all ONEs, the true contents or the complement (ONEs complement) of the contents of the respective bits of the B register.

The MIR buffers information being written to main system memory or a peripheral device. It is loaded from the barrel switch output and its output is sent to the Switch Interlock, or to the B register.

The adder in the LU is a modified version of a straightforward carry look-ahead adder. Inputs to the adder are from selection gates which allow various combinations of the A, B and Z inputs. The A input is from the A register output selection gates and the B input is from the B register true complement selection gates. The Z input is an external input to the LU and can be:

1. The output of the counter in the MCU into the most significant eight bits with all other bits being ZEROs.
2. The output of the literal register in the MCU into the least significant eight bits with all other bits being ZEROs.
3. An optional input (depending upon the word length) into the middle bytes (which only exists in Interpreters that have word lengths greater than 16 bits) with the most and least significant bytes being ZEROs.
4. The output of the AMPCR in the MCU into the least significant 12 bits with all other bits being ZEROs.
5. ALL ZEROs.

Using various combinations of inputs to the selection gates, any two of the three inputs can be added together, or can be added together with an additional ONE added to the least significant bit. Also, all binary Boolean operations between any two adder inputs can be performed.

The barrel switch is a matrix of gates that shifts a parallel input data word any number of places to the left or right, either end-off or end-around.

The output of the barrel switch is connected to:

1. The A registers (A1, A2, A3)
2. The B register
3. Memory Information Register (MIR)
4. Least significant 16 bits of MCU (registers BR1, BR2, MAR, AMPCR, CTR)
5. Least significant three to six bits to CU (depending on word length) for the shift amount register (SAR).

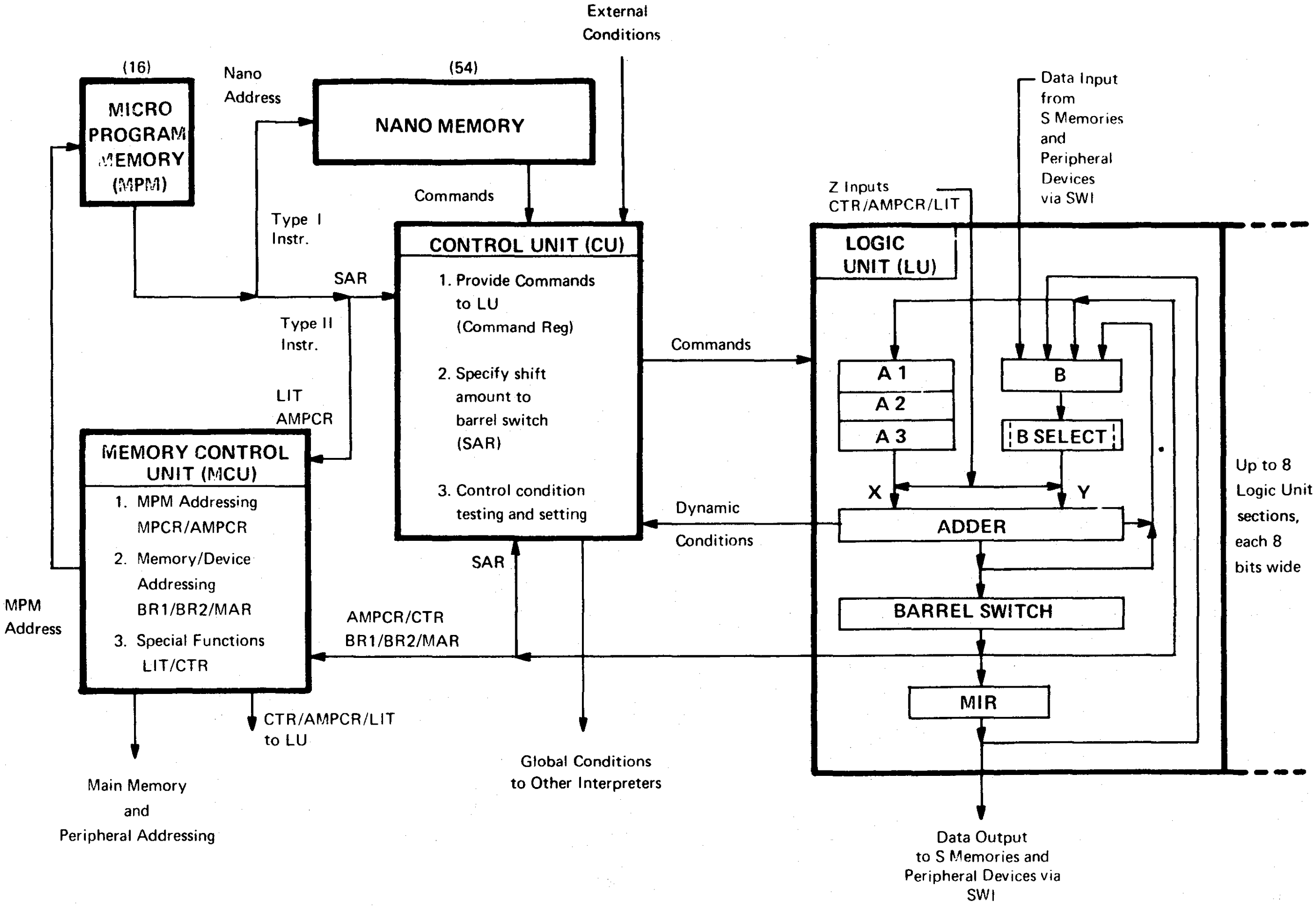

External Conditions
(16)
MICRO PROGRAM MEMORY (MPM)
Nano Address
(54)
NANO MEMORY
Commands
Type I Instr.
SAR
Type II Instr.
LIT
AMPCR
CONTROL UNIT (CU)
1. Provide Commands to LU (Command Reg)
2. Specify shift amount to barrel switch (SAR)
3. Control condition testing and setting
MEMORY CONTROL UNIT (MCU)
1. MPM Addressing MPCR/AMPCR
2. Memory/Device Addressing BR1/BR2/MAR
3. Special Functions LIT/CTR
MPM Address
SAR
AMPCR/CTR
BR1/BR2/MAR
CTR/AMPCR/LIT to LU
Main Memory and Peripheral Addressing
Global Conditions to Other Interpreters
Commands
Dynamic Conditions
Z Inputs CTR/AMPCR/LIT
Data Input from S Memories and Peripheral Devices via SWI
LOGIC UNIT (LU)
A 1
A 2
A 3
B
B SELECT
X
Y
ADDER
BARREL SWITCH
MIR
Up to 8 Logic Unit sections, each 8 bits wide
Data Output to S Memories and Peripheral Devices via SWI

Figure 3—Interpreter block diagram

Control unit (*CU*)

This unit has five major sections (shown in Figure 4): the shift amount register (SAR), the condition register (COND), part of the control register (CR), the MPM content decoding and the clock control.

The functions of the SAR and its associated logic are:

1. To load shift amounts into the SAR to be used in the shifting operations.
2. To generate the required controls for the barrel switch to perform the shift operation indicated by the controls from the Nanomemory.
3. To generate the "word length complement" of the SAR contents, where the "complement" is defined as the amount that will restore the bits of a word to their original position after an end-around shift of N followed by an end-around of the "complement" of N.

The condition register (COND) section of the CU performs four major functions:

1. Stores 12 resettable condition bits in the condition register. The 12 bits of the condition register are used as error indicators, interrupts, status indicators, and lockout indicators.
2. Selects 1 of 16 condition bits; 12 from the condition register and 4 dynamically generated during the present clock time (by the previous instruction being completed) in the Logic Unit for use in performing conditional operations.
3. Decodes bits from the memory for resetting, setting or requesting the setting of certain bits in the condition register.
4. Resolves priority among Interpreters in the setting of global condition (GC) bits which provides a facility for inter-Interpreter lock-out control.

The control register is a 36-bit register that stores the

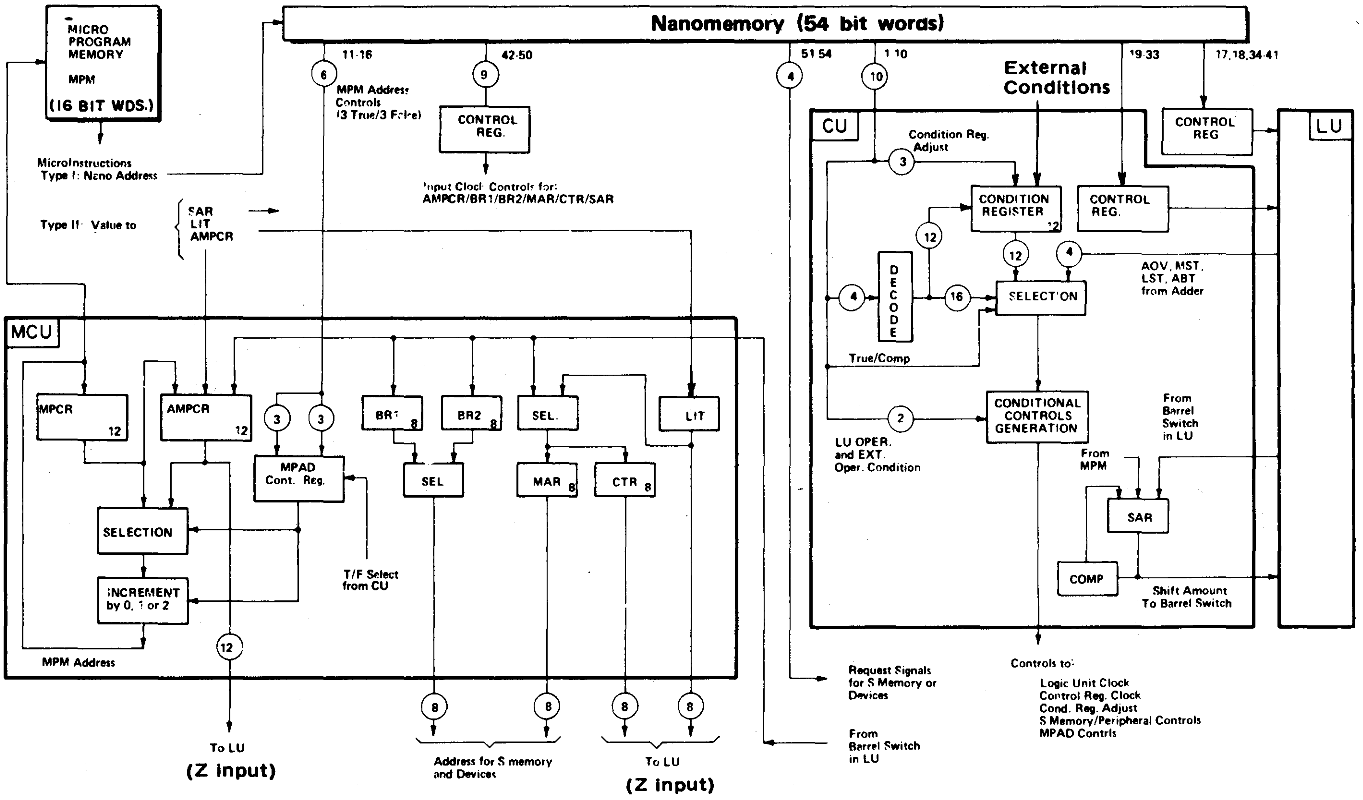


Figure 4—MCU/CU block diagram

subset of the control signals from the memory that are used in the LU, CU and MCU for controlling the execution phase of a microinstruction.

The MPM content decoding determines the use of the MPM as a Type I instruction (address) or as a Type II instruction (literal) based upon the first four bits of the MPM. Several decoding options are available.

Memory control unit (MCU)

One MCU is required for an Interpreter (providing access to 2^{16} words maximum) but a second MCU may be added to provide additional memory addressing capability (2^{32} words maximum). This unit has three major sections (shown in Figure 4):

1. The microprogram address section contains a microprogram count register (MPCR), the alternate microprogram count register (AMPCR), the incrementer, the microprogram address controls register, and their associated control logic. This section is used to address the MPM for the sequencing of the microinstructions. The AMPCR content may be used as input to the adder.
2. The memory/device address section contains the memory address register (MAR), base registers one and two (BR1, BR2), the output selection gates, and the associated control logic.
3. The Z register section contains registers which are the Z inputs to the LU adder: a loadable counter (CTR), the literal register (LIT), selection gates for the loadable counter and their associated control logic.

Interpreter operation

The operation of the Interpreter is described assuming a split MPM (16 bits)/Nano (54 bits) memory scheme (see Figure 5). During each clock, a microinstruction is read from the MPM. The first few bits of this microinstruction indicate which of two types of instruction it is. If it is a Type I instruction, the remaining bits of the MPM word specify a Nanomemory address to be accessed. The Nanomemory is then initiated and its output, a set of 54 bits, provides the control functions as indicated in Table II.

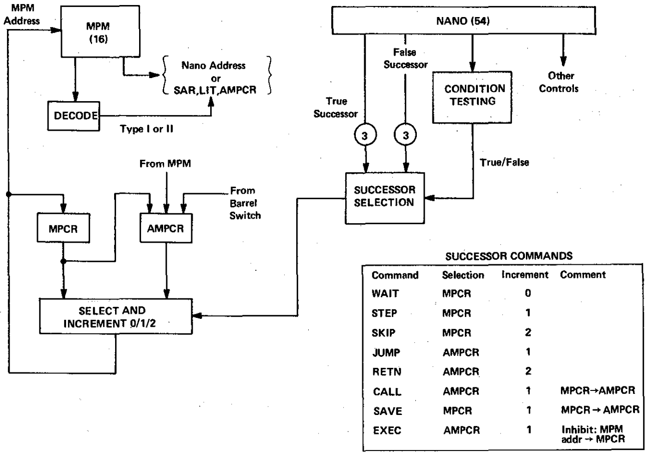
If the microinstruction is Type II (again specified by the first few bits of the MPM word), the remaining bits of the MPM word are stored into one of three registers; namely, the SAR, LIT, or the AMPCR. The determination of which register is to be loaded is also specified by the first bits of the MPM word. The Nanomemory is not accessed during a Type II operation. A Type II microinstruction has an implicit STEP successor.

The sequencing of microprogram instructions is controlled by the following procedure: The Nanomemory provides information to the condition testing logic indicating which condition is to be tested. The condition testing logic provides a True/False signal to the successor selection logic which selects between the three True and three False successor bits (also from the Nanomemory). The three selected bits (True/False) provide eight possible successor command combinations also shown in Figure 5.

The eight successor commands and their associated interpretations are:

Wait	Repeat the current instruction
Step	Step to the next instruction
Skip	Skip the next instruction
Jump	Jump to another area of MPM (as specified by AMPCR)
Retn	Return from Micro subroutine
Call	Call a Micro subroutine
Save	Save the address of the head of a loop
Exec	Execute one instruction out of sequence

The particular chosen successor command then provides controls used in the selection (MPCR/AMPCR) and incrementing logic which generates the next MPM address. Except for the EXEC command the MPCR is loaded with this MPM address.



SUCCESSOR COMMANDS

Command	Selection	Increment	Comment
WAIT	MPCR	0	
STEP	MPCR	1	
SKIP	MPCR	2	
JUMP	AMPCR	1	
RETN	AMPCR	2	
CALL	AMPCR	1	MPCR→AMPCR
SAVE	MPCR	1	MPCR→AMPCR
EXEC	AMPCR	1	Inhibit: MPM addr→MPCR

Figure 5—Microprogram instruction sequencing

TABLE II—Nanocodes

Nano-Bits	
1- 4	Select a condition
5	Selects true or complement of condition
6	Specifies conditional or unconditional LU operation
7	Specifies conditional or unconditional external operation (memory or device)
8-10	Specify set/reset of condition
11-16	Microprogram address controls (wait, skip, step, etc.) for true and false successor
17-26	Selects A, B, and/or Z adder inputs
27	Carry control
28-31	Select Boolean or basic arithmetic operations
32-33	Select shift operation
34-36	Select inputs to A registers
37-40	Select inputs to B register
41	Enables input to MIR
42	Enables input to AMPCR
43-48	Enable and select input to address registers and counter (MAR, BR1, BR2, CTR)
49-50	Select SAR preset
51-54	Select external operations (read, write, lock, etc.)

Each Type I microinstruction requires two clock intervals for its completion. The first (phase I) involves the fetching of the instruction from the MPM/Nanomemories and the second (Phase 3) executes the fetched instruction. Figure 6 illustrates these two basic phases (1 and 3) of each Type I microinstruction.

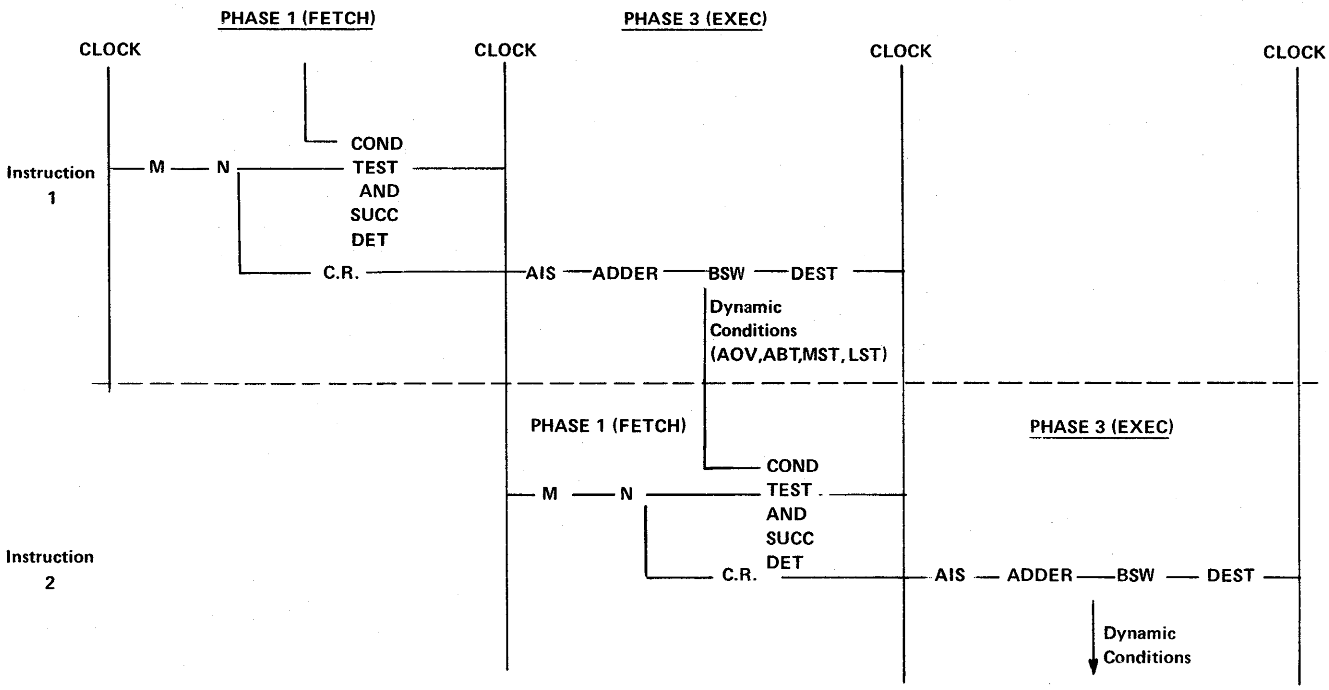
It should be noted that each phase 3 (execution) clock interval of instruction I is overlapped with phase 1 (fetch) clock interval of instruction I+1. Hence, the performance of each microinstruction requires effectively one clock interval.

As seen in Figure 6 the fetch phase involves: MPM memory accessing, Nanomemory accessing, the logic for condition testing and successor determination and in parallel the logic for loading the control register.

The execute phase includes adder input selections, adder function, barrel switch shifting of the adder output, and the destination register(s) loading. Figure 7 shows various timing examples for Type I and Type II instruction sequences.

INTERPRETER MICROPROGRAMMING

The pattern of 1s and 0s in the MPM and Nanomemories (together with the data) determine the operation of the Interpreter. The microprogrammer is concerned with the generation of these patterns to provide the desired control functions. However, instead of actually writing these patterns the microprogrammer is assisted by a microtranslator (or assembler) that allows



M = MPM ACCESS TIME
N = NANO ACCESS TIME
COND TEST AND SUCC DET. = CONDITION TEST AND SUCCESSOR DETERMINATION
BSW = BARREL SWITCH
DEST = BARREL SWITCH OUTPUT DESTINATIONS;I.E., REGISTERS (B, CTR, ETC.) AND THEIR INPUT LOGIC
C.R. = COMMAND REGISTER AND ASSOCIATED LOGIC
AIS = ADDER INPUT SELECTION FROM COMMAND REGISTER

Figure 6—Timing analysis, Type I instructions

him to write microinstructions mnemonically. The microtranslator then scans these instructions and produces the pattern of 1s and 0s to be placed into the MPM and Nanomemories.

Figure 8 indicates the ease with which one can learn to microprogram the machine and the simplicity of the microprogram structure. The high degree of parallelism extant in the Interpreter is also evident from the powerful statements that can be expressed. For example, the following actions may be expressed and performed in one instruction:

- test a condition (for either True or False)
- set/reset a condition
- initiate an external operation (e.g., memory read)
- perform an add operation
- shift the result of the add
- store the results in a number of registers
- increment a counter
- complement the shift amount
- choose the successor microinstruction

It is also possible to perform these operations conditionally or unconditionally as suggested in Figure 8. The group A and group B portions (either, neither, or both) of the microinstruction may be placed before the condition test portion of the instruction. This will result in that portion (A and/or B) being performed unconditionally.

The following four microinstruction examples illustrate both the parallelism and the conditional/unconditional properties of the microinstructions.

(1) If NOT LST then set LC1, MR1; A1+B+1C→A2, MIR, CSAR, INC; Step else Jump
(2) Set LC1, MR1; If NOT LST then A1+B+1C→A2, MIR, CSAR, INC; Step else Jump
(3) A1+B+1C→A2, MIR, CSAR, INC; If NOT LST then Set LC1, MR1; Step else Jump
(4) Set LC1, MR1; A1+B+1C→A2, MIR, CSAR, INC; if NOT LST then; Step else Jump

In (1) the LST bit is tested and if not true, the local

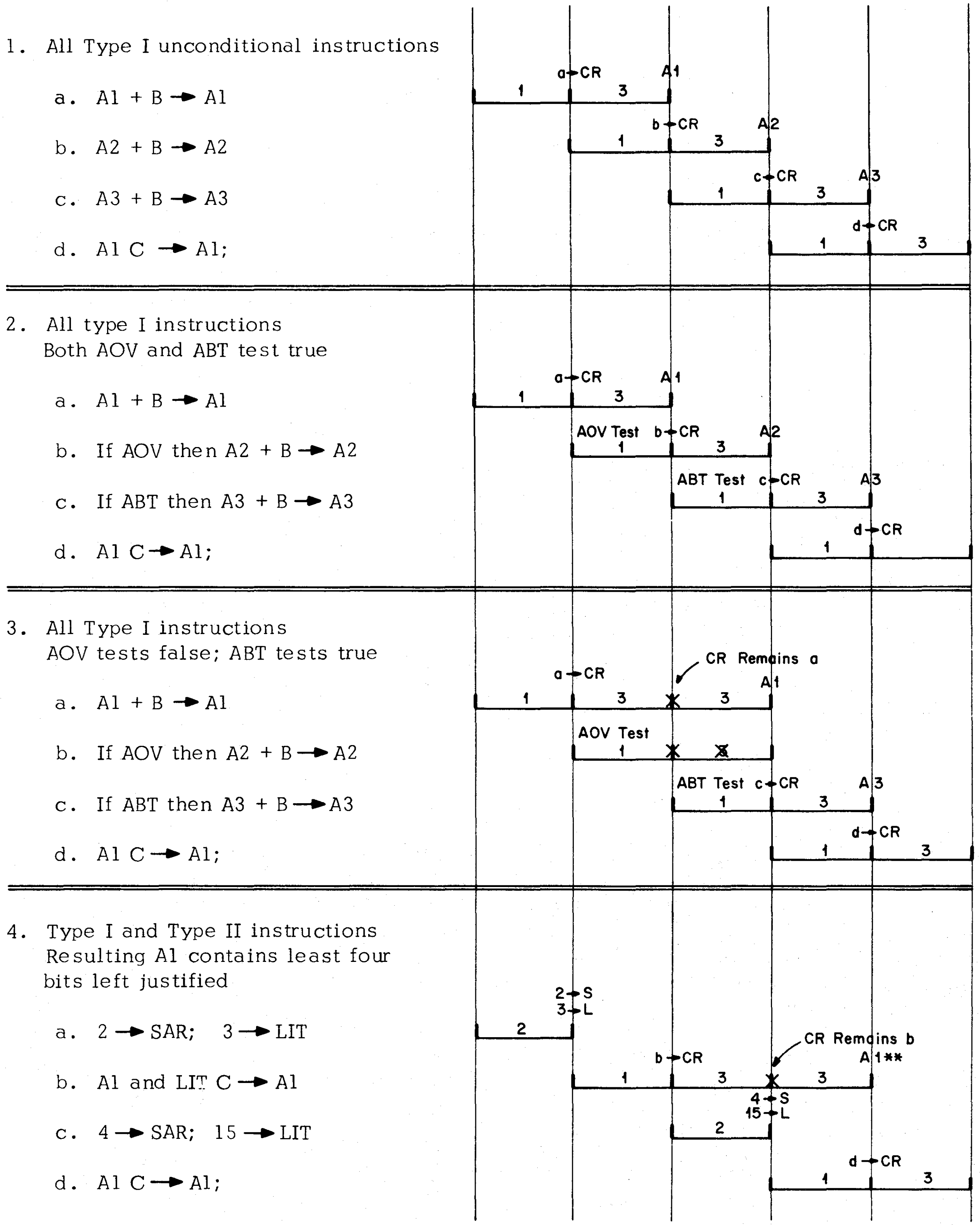


Figure 7—Timing examples

Type I — Use of nano memory (54 bits)

		A		B			
Nanobits	1-7 (7)	8-10 (3)	51-54 (4)	17-41 (25)	42-50 (9)	11-13 (3)	14-16 (3)
*; If	Condition then	Condition Adjust;	External	LU	MCU/CU;	True Succ	else False Succ
	GC1/2	Set LC1/2/3	Main	A/Z Select	Control for:	wait	wait
	LC1/2/3	Set GC1/2	Memory:	B/Z Select		step	step
	SAI	Set INT	Read/Write	Adder Function	AMPCR	save	save
	EX1/2/3	Reset GC		Shift Select	BR1/2	skip	skip
	MST		Device:	Destination(s)	MAR	jump	jump
	LST		Read/Write		CTR, INC	exec	exec
	ABT		Lock/Unlock		SAR, CSAR	call	call
	AOV					retn	retn
	COV						
	INT						
	RDC						

*Groups A and B may be executed either conditionally as shown or unconditionally by being placed before condition test.

Type II — Loads any of 3 specified registers (no nano memory access, step successor)
Four variations

k → SAR
k → LIT
k → AMPCR
k_1 → SAR; k_2→ LIT

Figure 8—Microinstruction types

condition 1 (LC1) is set, memory read is initiated (MR1), the function A1+B+1 is performed in the adder and the adder output is shifted circular and the result stored in both the A2 and MIR registers, the content of the shift amount register is complemented (CSAR), the counter is incremented (INC), and the true successor (STEP) is selected.

In (2) the LC1 is set and the memory read is initiated (MR1) unconditionally (i.e., without considering the LST bit). The remaining functions are conditionally performed as in (1).

In (3), the functions A1+B+1C→A2, MIR, CSAR, INC are performed unconditionally but set LC1 and MR1 are performed conditionally.

In (4) the functions Set LC1, MR1, A1+B+1C→ A2, MIR, CSAR, INC are all performed unconditionally and only the successors Step and Jump depend upon the LST test.

The Interpreter microprogramming reference card (Figure 9) specifies the use of each of the MPM and Nano bits and defines the meaning of the mnemonics found in the microprogram examples.

Two simple examples demonstrating the microprogramming of the Interpreter are shown in Figure 10 Binary Multiply and Figure 12 Fibonacci Series generation. The comments serve to explain the function of each microinstruction step. Figure 11 shows the microtranslator output (1 and 0 patterns for MPM and Nano) for the Binary Multiply example.

INTERPRETER APPLICATIONS

General

As mentioned earlier, one of the design objectives was to provide versatility so that a wide range of applications may be performed. The tasks to which the Interpreter has been successfully applied include peripheral controllers, emulators, high level language executors, and special function operators. This versatility allows the Interpreter to attack not only a wide range of problem areas but also every phase of a problem solution.

As indicated in Figure 13 a data processing facility receives programs written in either machine language or a high level language (e.g., Algol, Cobol, Fortran) and data upon which these programs operate. If the program is written in a high level language, then a compilation (or preprocessing) phase may be performed to translate this program into machine language form before execution. Each phase of this problem solution operation—recognition, code generation and execution—can be performed by a microprogrammed processor; in fact, each phase can be tuned to optimally perform the task.

Three main data processing functions to which the Interpreter may be applied are: Emulation of existing or hypothetical machines, "direct execution" of high level languages, and problem solution through microprogram "tuning" of the machine to the problem. These three function areas—emulation, "direct execution," problem "tuning"—will now be defined and the Interpreter approach to their implementation will be described.

Emulation

In a conventional (non-microprogrammable) computer, the control unit is designed using a fixed set of flip-flops and decoders to perform the execution phase of the operation. The execution phase includes the fetching of machine instructions, calculation of effective operand addresses, and the provision of each microstep command in the performance of the machine instruction.

Emulation is defined in this paper as the ability to execute machine language programs intended for one machine (the emulated machine) on another machine (the host machine). Within this broad definition, any machine with certain basic capabilities can emulate any other machine; however, a measure of the efficiency of operation is generally implied when the term emulation is used. For example, if a conventional computer

MICRO CONTROLS

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	ø	* SAR						ø	ø	ø	ø	ø	ø	ø	ø
1	0	* SAR						LIT							
1	1	0	ø	* AMPCR											
1	1	1	0	ø	ø	ø	ø	LIT							
1	1	1	1	* NANO ADDRESS											

ø Unused
* Shorter fields are right justified

NANO CONTROLS

Parentheses surround optional lexic units, provided by default.

1	2	3	4	Condition Tested Result is Boolean cnd
0	0	0	0	GC1
0	0	0	1	GC2
0	0	1	0	LC1
0	0	1	1	LC2
0	1	0	0	MST
0	1	0	1	LST
0	1	1	0	ABT
0	1	1	1	AOV
1	0	0	0	COV
1	0	0	1	SAI
1	0	1	0	RDC
1	0	1	1	LC3
1	1	0	0	EX1
1	1	0	1	INT
1	1	1	0	EX2
1	1	1	1	EX3

5	FT Condition Value	
0	NOT	$\overline{\text{cnd}}$=:SC
1	--	cnd=:SC

6	Logic Unit Conditional
0	Do Unconditionally
1	Do Conditionally if SC

7	Ext Op (MDOP/CAJ) Conditional
0	Do Unconditionally
1	Do Conditionally if SC

8	9	10	Condition Adjust -- CAJ
0	0	0	--
0	0	1	SET LC2
0	1	0	SET GC2
0	1	1	RESET GC
1	0	0	SET INT
1	0	1	SET LC3
1	1	0	SET GC1
1	1	1	SET LC1

11	12	13	Successor	14	15	16
Then Part Used if SC=1			to MPAD Ctls	Else Part Used if SC=0		
0	0	0	WAIT	0	0	0
0	0	1	(STEP)	0	0	1
0	1	0	SAVE	0	1	0
0	1	1	SKIP	0	1	1
1	0	0	JUMP	1	0	0
1	0	1	EXEC	1	0	1
1	1	0	CALL	1	1	0
1	1	1	RETN	1	1	1

17	18	19	Adder X Input
0	0	0	(0)
0	0	1	LIT
0	1	0	ZEXT
0	1	1	CTR
1	0	0	Z
1	0	1	A1
1	1	0	A2
1	1	1	A3

20	21	22	23	24	25	26	Adder Y Input
0	0	-	-	-	-	-	B0--
0	1	-	-	-	-	-	BT--
1	0	-	-	-	-	-	BF--
1	1	-	-	-	-	-	B1--
-	-	0	0	0	-	-	B-0-
-	-	1	0	0	-	-	B-T-
-	-	-	-	-	0	0	B--0
-	-	-	-	-	0	1	B--T
-	-	-	-	-	1	0	B--F
-	-	-	-	-	1	1	B--1
Comp		1	0	0	Comp		B-F-*
Comp		0	0	0	Comp		B-1-*
0	0	0	0	1	0	1	LIT
0	0	0	1	0	0	0	ZEXT
0	1	0	1	1	0	0	CTR
0	1	1	0	1	0	1	Z
0	0	1	1	0	0	1	AMPCR

*Use Adder Operation with Complement Y

27	Inhibit Carries into Bytes	
0	--	Allow
1	IC	Inhibit

28	29	30	31	Adder Operation	Logic
0	0	0	0	X + Y	
0	0	0	1	X NOR Y	$\overline{X}\ \overline{Y}$
0	0	1	0	X NRI Y	$\overline{X}\ Y$
0	0	1	1	X + Y + 1	
0	1	0	0	X NAN Y	$\overline{X} \vee \overline{Y}$
0	1	0	1	X OAD Y	X + (X v Y)
0	1	1	0	X XOR Y	$X\ \overline{Y} \vee \overline{X}\ Y$
0	1	1	1	X NIM Y	$X\ \overline{Y}$
1	0	0	0	X IMP Y	$\overline{X} \vee Y$
1	0	0	1	X EQV Y	$X\ Y \vee \overline{X}\ \overline{Y}$
1	0	1	0	X AAD Y	X + (XY)
1	0	1	1	X AND Y	X Y
1	1	0	0	X - Y - 1	$X + \overline{Y}$
1	1	0	1	X RIM Y	$X \vee \overline{Y}$
1	1	1	0	X OR Y	X v Y
1	1	1	1	X - Y	$X + \overline{Y} + 1$

32	33	Shift Type Selection for BSW	
0	0	--	No Shift
0	1	R	Right End Off
1	0	L	Left End Off
1	1	C	Right Circular

34	35	36	A Register Input from BSW	
0	0	0	--	No Change
1	-	-	A1	
-	1	-	A2	
-	-	1	A3	

37	38	39	40	B Register Input Select	
0	0	0	0	--	No Change
0	0	0	1	BC4	Comp 4 Bit Carries
1	0	0	0	BAD	Adder
1	0	0	1	BC8	Comp 8 Bit Carries
1	0	1	0	BBA	BSW v Adder
1	0	1	1	B	BSW
1	1	0	0	BEX	External
1	1	0	1	BMI	MIR
1	1	1	0	BBE	BSW v External
1	1	1	1	BBI	BSW v MIR

41	MIR Input from BSW	
0	--	No Change
1	MIR	

42	AMPCR Input from BSW	
0	--	No Change
1	AMPCR	

43	44	45	46	Mem Dev Address Input	
0	0	0	-	--	No Change
-	-	1	0	LMAR	From LIT
-	-	1	1	MAR	From BSW
-	1	0	-	BR2	From BSW
-	1	1	1	MAR2	From BSW
1	-	0	-	BR1	From BSW
1	-	1	1	MAR1	From BSW

46	47	48	Counter Input	
-	0	0	--	No Change
0	0	1	LCTR	From LIT*
1	0	1	CTR	From BSW*
-	1	0	INC	+1

*Ones Complement

49	50	SAR Input	
0	0	--	No Change
0	1	CSAR	Complement
1	0	SAR	From BSW

51	52	53	54	Mem Dev Op--MDOP	
0	0	0	0	--	No Change
0	0	1	0	MR1	
0	0	1	1	MR2	
0	1	1	0	MW1	
0	1	1	1	MW2	
1	0	0	0	DL1	
1	0	0	1	DL2	
1	0	1	1	DR1	
1	0	1	1	DR2	
1	1	0	0	DU1	
1	1	0	1	DU2	
1	1	1	0	DW1	
1	1	1	1	DW2	

67068

Figure 9—Interpreter microprogramming reference card

```
Assumptions

    (1)  Sign-magnitude number representation

    (2)  Multiplier in A3; multiplicand in B

    (3)  Double length product required with resulting
         most significant part, with sign, in B and least
         significant part in A3

1.    A3 XOR B→ ;if LC1

2.    $B_{0TT}$→ A2; if MST then Set LC1

      Comment: Step 1 resets LC1. Steps 1 and conditional part of 2
      check signs; if different, LC1 is set.

3.    $B_{000}$→B, LCTR

      Comment: Steps 2 and 3 transfer multiplicand (0 sign) to A2
      and clear B.

4.    "N"→LIT; 1→SAR

      Comment: Steps 3 and 4 load the counter with the number
      (N = magnitude length) to be used in terminating the multiply
      loop and load the shift amount register with 1.

5.    A3 R→A3; Save

      Comment: Begins test at least bit of multiplier and sets up loop.

6.    LOOP: If not LST then $B_{0TT}$C→B skip else step

7.    A2 + $B_{0TT}$C→B

8.    A3 OR $B_{T00}$R→A3, INC; if not COV then jump else step

      Comment: 6 through 8 - inner loop of multiply (average 2.5
      clocks/bit).

9.    If not LC1 then $B_{0TT}$ → B; skip else step

10.   $B_{1TT}$→B

      Comment: If LC1 = 0, the signs were the same, hence force sign bit
      of result in B to be a 0.
```

Figure 10—Example of microprogram for binary multiply

emulates another computer, the emulation efficiency is generally very poor since for each machine language instruction of the emulated machine there corresponds a sequence of machine instructions to be executed on the host machine. This relatively inefficient operation for conventional computers with fixed machine instructions (called simulation in IBM literature—see Husson[25] and Tucker[17]) turns out to be significantly more efficient on microprogrammable computers. In a microprogrammed machine, the controls for performing the fetching and execution of the machine instructions of the emulated machine consist of a sequence of microinstructions which allows the increased efficiency.

There are a number of attributes of the Interpreter which make it uniquely appropriate for emulation applications. In addition to the obvious flexibility allowed by microprogramming, the following features of the Interpreter contribute significantly to an effective emulation capability:

1. Modularly variable logic unit word length permits configuring the emulator machine for most effective match with the emulated machine.
2. The barrel switch provides for arbitrary length shifts in a single clock time, a very effective facility for instruction "break-apart" and address calculation.
3. Powerful and extremely flexible microinstructions. There is no fixed, limited-size instruction repertoire; rather, the Interpreter's nanoinstruction provides practically unlimited flexibility in instruction composition. The microinstructions of some machines have a rather elemental capability; here, it is possible to compose microinstructions of relatively great power. A single micro-nanoinstruction may specify selective condition testing and setting, conditional or unconditional logic unit operations between selected registers, optional shifting, literal loading of certain registers, and independently conditional or unconditional memory or device read-write operations.
4. The zoned B-register selection affords convenient testing and setting of special bits.
5. Condition setting and testing allows ease of communication with peripherals and memories.
6. Multiple units for a variety of system functions and multiprocessing may be interconnected via

```
 000   PROGRAM BIMULT;
 100   A3 XOR  B = :  ; IF  LC1;
 200   B0TT = : A2; IF MST THEN SET LC1;
 300   B000  =  :  B, LCTR;
 400   N = : LIT; 1 = : SAR;
 500   A3  R  =  :  A3; SAVE;
 600   LOOP:  IF NOT LST THEN B00T  C = : B; SKIP ELSE STEP;
 700   A2 + B0TT C  = : B;
 800   A3 OR BT00  R  =  : A3, INC; IF NOT COV THEN JUMP ELSE STEP;
 900   IF NOT LC1 THEN B0TT = : B; SKIP ELSE STEP;
1000   B1TT = : B;
```

```
 1          NANO ADDRESS=             0         1111  000000000000
       3    5   13    16    17    18   19    21    22    26     29     30
 2          NANO ADDRESS=             1         1111  000000000001
       2    7    8     9    10    13   16    22    26    35
 3          NANO ADDRESS=             2         1111  000000000010
       2    5   13    16    37    39   40    48
 4          SAR= 1          LIT = 0             10  00000100000000

 5          NANO ADDRESS=             3         1111  000000000011
       2    5   12    15    17    18   19    33     36
 6          NANO ADDRESS=             4         1111  000000000100
       2    4    5     6    12    13   16    22     26   32     33     37    39    40
 7          NANO ADDRESS=             5         1111  000000000101
       2    5   13    16    17    18   22    26     32   33     37     39    40
 8          NANO ADDRESS=             6         1111  000000000110
       1   11   16    17    18    19   21    28    29    30     33     36    47
 9          NANO ADDRESS=             7         1111  000000000111
       3    6   12    13    16    22   26    37    39    40
10          NANO ADDRESS=             8         1111  000000001000
       2    5   13    16    20    21   22    26    37    39     40
```

Figure 11—Example of microtranslator output

Assumptions:

A1 contains starting address for storing of series

A2 contains the number representing the length of the series to be computed

1. A1 → MAR1

   Comment: Load starting address of series into address register

2. B_{000} → B, MIR

3. B_{001} → A3; MW1

   Comment: Load initial element of series (0) into A3 and MIR and write it into starting address. Load second element of series (1) into B.

4. A2 → CTR;SAVE

   Comment: Load counter with length of series; the counter will be incremented for each generation of an element of the series; COV will signify completion. The SAVE sets up the loop.

5. LOOP: If SAI then A1 + 1 → A1, MAR1, INC, Step else Wait

   Comment: Set up the next address and increment counter

6. A3 + B → MIR

   Comment: Generate new element in series and place in MIR

7. B → A3; BMI, MW1; If NOT COV then Jump else Step

   Comment: Write new element into next address
   Transfer i − 1 element to A3
   Transfer i element to B
   Test counter overflow for completion (go to LOOP, if not done)

8. DONE:

Figure 12—Example of microprogram for generation of Fibonacci series

the Switch Interlock to facilitate emulation of a large range of target machine structures.

These characteristics of the Interpreter that prove so useful in emulation derive from the basic design concept that the Interpreter was to be both simple enough and yet powerful enough that one basic element could be microprogram specialized and modularly expanded

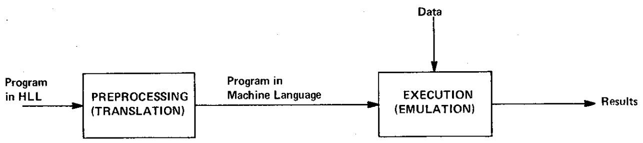
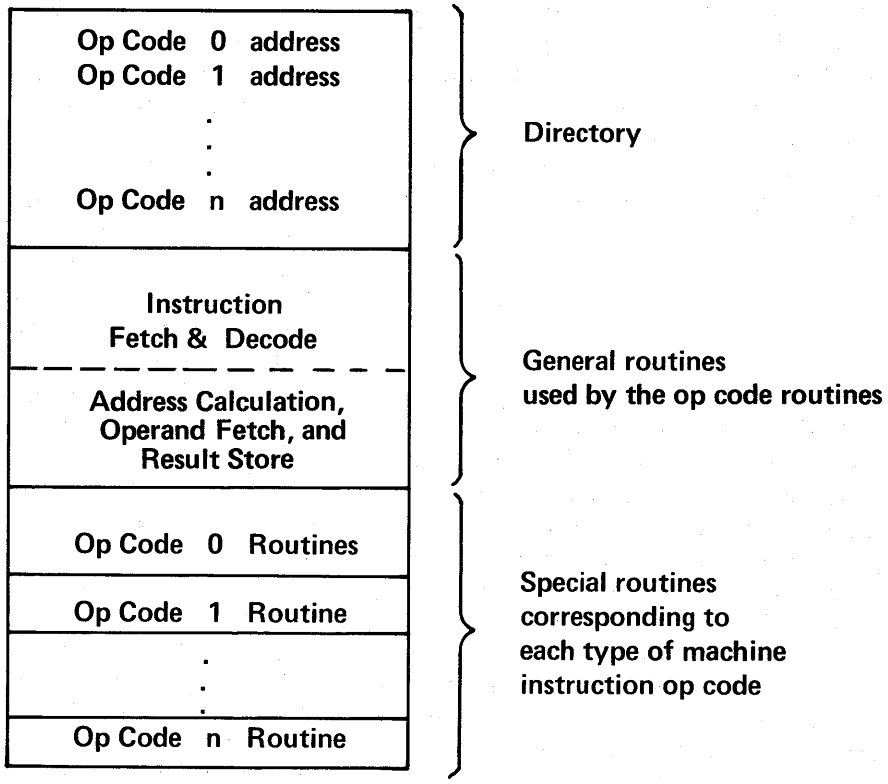


Figure 13—Basic data processing functions


Op Code 0 address
Op Code 1 address
Op Code n address
Directory
Instruction Fetch & Decode
Address Calculation, Operand Fetch, and Result Store
General routines used by the op code routines
Op Code 0 Routines
Op Code 1 Routine
Op Code n Routine
Special routines corresponding to each type of machine instruction op code

General operating procedure

1. **Instruction fetch - fetches next machine instruction to be interpreted**
2. **Instruction decode - separates the op code field in the machine instruction.**
3. **Pass control to appropriate op code routine - using the Directory as a pointer.**
4. **Execute the specified op code routine - using the general routines (esp. addressing) as needed.**
5. **Return control to the instruction fetch routine**

Figure 14—Emulation—MPM map and operating procedure

to efficiently cover a broad range of applications from simple controllers to complex processors or computer systems.

It is important to realize that the machine being emulated may be either an existing or a hypothetical machine. The basic items necessary to define a machine and hence to emulate it are:

1. memory structure (registers, stacks, etc.),
2. machine language format (0, 1, 2, 3 address) including effective operand address calculation, and
3. operation codes and their functional meaning (instructions for arithmetic, branching, etc.).

The process of writing an emulation therefore, involves the following analyses and the microprogramming of these basic items:

1. Mapping the registers (stacks, etc.) from the emulated machine onto the host machine,
2. Analysis of the machine language format and addressing structure,
3. Analysis of each operation code defined in the machine language.

All the registers of the emulated machine must be mapped onto the host machine; that is, each register must have a corresponding register on the host machine. The most frequently used registers are mapped onto the registers within the Interpreter Logic Unit (e.g., registers A1, A2, A3). The remaining registers are stored either in the main system memory or in special high speed registers depending on the desired emulation speed. The machine language format may be 0, 1, 2, 3 address or variable length for increased code density, and may involve indexing, indirection, relative addressing, stacks, and complex address conversions. Figure 14 shows the general microprogram requirements (MPM Map) and operating procedure for the emulation task.

Direct execution

The term "direct execution" has a variety of meanings resulting from the possible divisions of effort in the preprocessing and execution phases of the data processing operation. It may be understood that a machine "directly executes" a high level language (HLL) if a program written in that HLL is submitted to the machine and results occur without regard to how this has been accomplished. Under this broad definition, any machine that has a compiler for this HLL would be considered a "direct executor" of that HLL and this represents one extreme. The amount of preprocessing in this case is quite significant since an entire compiler is required before execution begins.

At the other extreme, practically no preprocessing occurs and the entire HLL program resides in the system (e.g., memory or disc) as it was written (i.e., one-to-one correspondence with the characters of the input code). Each character scanned in the input source string causes an appropriate action. Although this definition involves no significant preprocessing, it is probably only of academic interest because of its slow inefficient operation. It appears that some amount of preprocessing is desirable and that an optimum degree of preprocessing may be arrived at for each environment and/or application.

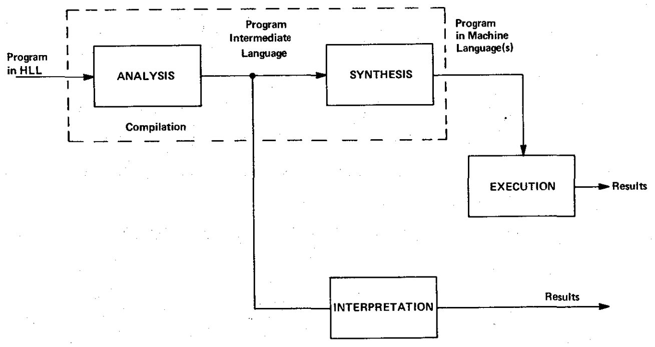


Figure 15—Direct execution of HLL

Figure 15 shows two approaches for arriving at results from a program written in a HLL. The one path involves the standard compilation process followed by an emulation and the other path involves interpretation.

Compilation consists of two basic functions: Analysis and Synthesis. The Analysis function depends upon the HLL and includes the scanner (of the input source string of characters) and the syntax recognizer which determines when a syntactic element of the HLL has been obtained. The output of this analysis is a program in intermediate language form which is usually in Polish postfix notion with tables indicating an internal representation of each element of the input string. This intermediate program has basically a one-to-one correspondence to the input HLL program.

The Synthesis function depends upon the machine that performs the execution phase and consists primarily of code generation. The input is the intermediate language program and the output is a program in the machine language of the execution machine. The code generation is generally a one-to-many mapping from intermediate to machine language.

The term Interpretation generally involves two functions—Analysis and Execution; that is, translation of a HLL program into an intermediate (or internal) form and execution of the program directly in this intermediate form without a translation to a machine language. The Analysis is basically the same as that described above for the compilation process. The meaning of the term Interpretation will be limited in the following discussion to the second part; that is, the execution of the program in its intermediate form.

An Interpretation is often slower in execution time than the equivalent execution of a machine language program. This might not always be the case in a microprogrammed environment, when the control structure of the HLL is significantly different from that of the host machine language, or when a significant portion of the execution time is spent in standard system routines. Interpretation does provide a better facility for development of new programs through built-in functions (e.g., debugging tools).

The functions involved with "direct execution"—Analysis, Synthesis, Interpretation—may all be more effectively performed in a microprogrammed environment. For each function, a set of microprograms may be developed to provide a significant increase in operating speed (see Tucker and Flynn[26]).

A combination of approaches is also easily implemented on a microprogrammed machine. The Interpretation approach may be used during the program development phase and depending upon its operating performance, it may be used for the repeated "production" phase. The compilation approach may be desired for the "production" phase if the execution of a machine language program is of higher performance than the Interpretation approach. The microprogrammed machine provides great versatility here also. In addition to providing a highly efficient compilation process, the machine code output may be modified until an "optimum" machine is defined for the application and/or high level language. This machine would then be emulated on the same Interpreter as described earlier ("Emulation").

The designing of conventional machines to match a particular high level language has been demonstrated to provide high performance. Two examples of this are the Burroughs B 5500 for the ALGOL language and the B 3500 for the COBOL language. A number of the HLL features are reflected in these architectures; for examples: The B 5500 run time stack in which the calling environment resides matches the ALGOL block structure and recursive procedures.

The B 5500 code stream is a reverse Polish notation corresponding to order of expression execution with intermixed operators, literals, operand calls, and descriptor calls. This not only matches the ALGOL operator precedence but also results in increased code density.

The B 5500 operand fetch is driven by data descriptor so that uniform codes can be used for call by name, value, and reference parameters.

The B 5500 performs arithmetic on real numbers, treating integers as zero exponent reals and Booleans as a real 0 or 1 thus providing uniform data representation.

The B 5500 stores arrays as a hierarchy of vectors that contain bases of the next lower level of arrays thus providing for the ALGOL N-dimensional dynamic arrays.

The B 3500 operators work on packed, unpacked, and mixed stored forms of decimal or character data thereby providing convenient management of COBOL variable length operands. The COBOL editing and moving operations are provided for by the primitive B 3500 instructions for character string editing and moving.

The ability to structure a machine for a particular language is enhanced through microprogramming; hence, these same capabilities may be more easily implemented than on the conventional machine.

Tuning the machine for the problem

In the past, conventional machines with fixed instruction sets were used with a force fit approach to solve a given problem. Furthermore, the language and computers (including instruction sets) were developed independently and were based on different requirements and constraints. The language design depended on the user and application and the computer hardware depended primarily on cost and general performance relative to all potential problems. At the juncture, a sometimes complex preprocessor (compiler) was required to make compatible the two designs.

Microprogrammed machines offer the unique opportunity to modify the system architecture to optimally solve a given problem. Languages may still be designed with the application and user in mind but the computer now may also be designed to be compatible with the application and the language. As a result, (1) the language is ideal for the user and application; (2) the preprocessing is significantly simpler because both ends (i.e., language and machine) are flexible, and (3) the machine provides an optimum solution to the problem by matching the language and application requirements.

Among the many advantages to this tuning to the problem approach in addition to this increased system performance is the fast turn around time from problem statement to problem solution. Machine hardware does not have to be built for each new application since the hardware is off-the-shelf and only the microprograms must be developed. This also allows a delayed binding of the required system specification until the problem is clearly defined. As a matter of fact, no permanent binding is ever required since the microprograms may be modified (expanded/contracted) at any time to meet new system needs.

Two basic methods for initiating the implementation of this tuning to the problem approach are: (1) design on paper the ideal machine and/or language for the particular task and then emulate that machine and/or directly execute the language and (2) use an existing machine and/or language as the starting point and emulate the machine or directly execute the language. In either case, the process of tuning is generally iterative in nature. The application programs may be executed using monitoring techniques to detect and gather a variety of statistics about the program. This monitoring is very easily implemented on a microprogrammed machine and may be used for examples (1) to analyze the operation code usage and (2) to determine the slow areas within the program where most of the time of execution is spent. This information may then be used to modify the instruction set by adding or deleting operation codes to the original set. In addition to the instruction sets that may be modified to optimize performance, the data may also be specified as to data types, word lengths, etc., depending on the data structures and accuracies required. The machine structure may also be modified (e.g., adding stacks or general

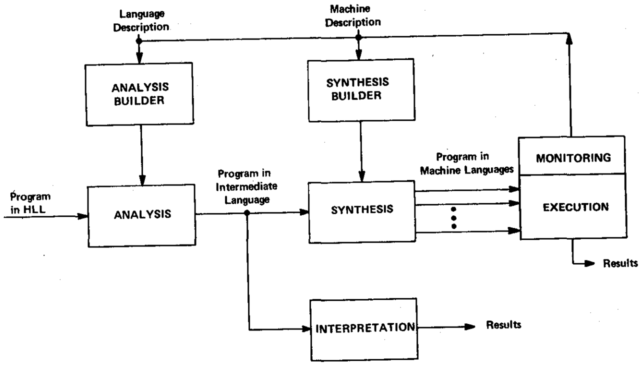


Figure 16—Tuning the machine for the problem

registers, indexing) and the machine language instruction format altered.

Figure 16 indicates a method used for tuning to the problem. The language description is entered into the Analysis Builder and the machine description is entered into the Synthesis Builder. The function of the Analysis and Synthesis Builders is to develop the internal tables (syntax recognition, etc.), machine code segments, etc., for use by the Analysis and Synthesis sections. This building function is presently performed manually but may eventually be performed with computer assistance. The functions of the Analysis, Synthesis, Interpretation and Execution sections are basically the same as previously described ("Direct Execution"). The prime enhancement here is the Monitoring of the Execution function and the feed-back of information for the language and machine descriptions. This allows iteration on the language and machine descriptions until an optimum solution is attained. The result is a language and machine ideal to solve the given problem. Any number of different machines may be emulated by simply modifying the microprogram set.

As an example of this process, let us assume that an emulation of some existing or hypothetical machine (simple or complex) exists. The application program may be run and statistics gathered (by monitoring) concerning this program. The monitoring for dynamic operation code usage, for example, requires only about 10 lines of microcode. The gathered statistics can indicate the execution areas where most of the processing time is spent; that is, the section(s) of machine language instructions most used. The function of this section(s) of machine instructions may then be specified, a single new machine instruction may be defined to perform this same function, and microcode may be developed to emulate this new machine instruction. This process of enhancing the original machine language allows a reduction in main memory requirements for the application program in addition to increasing the performance by a factor of 10 to 20. The estimated speed improvements were stated by Tucker and Flynn[26] in their comparison of the IBM360/50 and their suggested microprogrammed machine. The examples that they chose for demonstrating the increase in performance were: array element address calculation, Fibonacci series generation, and a table search algorithm. Interpreter analyses of these and other examples tend to verify these order of magnitude performance improvements.

REFERENCES

1 M V WILKES
The best way to design an automatic calculation machine
Manchester University Computer Inaugural Conference Proceeding 1951 p 16

2 W L VAN DER POEL
Micro-programming and trickology
John Wiley and Sons Inc 1962 Digital Information Processors

3 H T GLANTZ
A note on microprogramming
Journal ACM 3 Vol No 2 1956 p 77

4 R J MERCER
Micro-programming
Journal ACM 4 Vol No 2 1957 p 157

5 J V BLANKENBAKER
Logically microprogrammed computers
IRI Prof Group on Elec Com December 1958 Vol EC-7 No 2 p 103-109

6 G P DINEEN I L LEBOW et al
The logical design of CG24
Proc EJCC December 1958 p 91-94

7 T W KAMPE
The design of a general-purpose microprogram-controlled computer with elementary structure
IRE Trans June 1960 Vol EC-9 No 2 p 208-213

8 L BECK F KEELER
The C-8401 data processor
February 1964 Datamation p 33-35

9 O BOUTWELL JR
The PB 440 computer
February 1964 Datamation p 30-32

10 L D AMDAHL
Microprogramming and stored logic
February 1964 Datamation p 24-26

11 R H HILL
Stored logic programming and applications
February 1964 Datamation p 36-39

12 W C McGEE
The TRW-133 computer
February 1964 Datamation p 27-29

13 W Y STEVENS
The structure of SYSTEM/360 Part II—System implementation
IBM Systems Journal Vol 3 No 2 1964 p 136-143

14 A J MELBOURNE J M PUGMIRE et al
A small computer for the direct processing of Fortran statements
Computer Journ England April 1965 Vol 8 No 1 p 24-27

15 W C McGEE H E PETERSON

Microprogram control for the experimental sciences
Proc AFIPS 1965 FJCC Vol 27 pp 77-91

16 J GREEN
Microprogramming emulators and programming languages
Comm of ACM March 1966 Vol 9 No 3 pp 230-232

17 S G TUCKER
Emulation of large systems
Communications of the ACM December 1965 Vol 8 No 12 pp 753-761

18 A OPLER
Fourth-generation software, the realignment
Datamation January 1967 Vol 13 No 1 pp 22-24

19 I T HAWRYSZKIEWYCZ
Microprogrammed control in problem-oriented languages
IEEE Transactions on Electronic Computers October 1967 Vol EC-16 No 5 pp 652-658

20 G A ROSE
Integraphic, a microprogrammed graphical-interface computer
IEEE Transactions December 1967 Vol EC-16 No 6 pp 776-784

21 H W LAWSON
Programming language-oriented instruction streams
IEEE Transactions 1968 C-17 p 476

22 M V WILKES
The growth of interest in microprogramming—A literature survey
Comp Surveys Vol 1 No 3 Sept 1969 pp 139-145

23 R F ROSIN
Contemporary concepts of microprogramming and emulation
Comp Surveys Vol 1 No 4 Dec 1969 pp 197-212

24 L L RAKOCZI
The computer-sithin-a-computer: A fourth generation concept
Computer Group News Vol 2 No 8 March 1969 pp 14-20

25 S HUSSON
Microprogramming: Principles and practices
Prentice Hall Englewood Cliffs NJ 1970

26 A B TUCKER M J FLYNN
Dynamic microprogramming: Processor organization and programming
CACM April 1971 Vol 14 No 4 pp 240-250

Modeling, measurement and computer power*

by G. ESTRIN, R. R. MUNTZ and R. C. UZGALIS

University of California
Los Angeles, California

INTRODUCTION

Since the early 1960s the literature[9,32] reveals increasing concern with effectiveness of information processing systems and our ability to predict influences of system parameters. A recent survey paper[38] discusses methods of performance evaluation related to three practical goals: selection of the best among several existing systems; design of a not-yet existing system; and analysis of an existing accessible system. The classification of goals is useful, but we can point to neither the models nor the measures nor the measurement tools to allow reliable judgments with respect to those three important goals at this time.

We choose to discuss three issues which do not fall cleanly into Lucas' categories but which are certain to influence our ability to evaluate computer systems in the 1970s. The three issues are: effectiveness of models of computer systems; requirements to be met by measurement experiments; and application of modeling and measurement to the user interface with computer systems.

The first section provides a context for the other sections by reviewing parameters which make computing systems more or less powerful. The second section gives a critique of the state of modeling. The third section characterizes measurement tools. The fourth section discusses the role of measurement at the user interface.

COMPUTER POWER

We consider a computer system to be composed of: a centralized hardware configuration; a set of terminals for entry and exit of user programs and data; an operating system; public programs and data bases; user programs and data; and users and user protocol for entry and exit.

There is no accepted measure for global power or performance of computer systems. There is even no accepted measure for computer cost. Only when a subsystem or subfunction is isolated does it become possible to determine key parameters. However, it is useful to hypothesize such measures and consider influences on them.

Let us, therefore, define a conceptual measure which we call computer system power, P, as a multivariate polynomial function whose coefficients are significance weights. We would, of course, like to have a set of orthogonal functions whose independent variables correspond to measurable parameters but that state of happiness is not apparently within reach. In an attempt to exemplify our philosophy, the authors discuss a set of variables which should influence P keeping in mind that derivation of a figure of merit would require dividing P by some measure of cost.

We intuitively *expect* computer system power to increase if the:

- execution time of any CPU instruction is decreased
- access time of any memory subsystem is decreased
- transfer rate to or from any memory subsystem is increased
- transmission rate of any buss structure is increased
- transfer rate to and from any input or output device is increased
- delay in resource availability is decreased
- error recovery time is decreased
- number of useful public programs is increased
- performance of any public program is increased
- access time to any public data base is decreased
- arrival, execution, and departure rates of user programs are increased
- execution time or resource requirement of any user program is decreased

* This research was supported by the National Science Foundation, Grant No. GJ 809.

- number of effective users increases
- amount of protocol for any user decreases

In a deeper, even more qualitative sense, we expect a computer system to be more powerful if the following conditions hold:

- system manager has a model permitting adaptation to changing load
- errors and system imbalances are reported to maintainers and developers
- program documentation and measurements permit modification with few side effects
- average number of user runs before correct execution is decreased
- the quality of any user program increases in the sense that there is more effective use of a source language on a given computer system.

Although the above observations are useful in stating expected events of concern they ignore interactions between such events and give no indication of weighted importance of the individual events. We further characterize our systems by the following simple remarks.

If the time required for every physical transition to reach its new stable state were halved, we would expect throughput of the system to double. If only some of the events were reduced in transition time, we could no longer guarantee that there would be a reduction in computation time because the scheduling of events is a function of starting and stopping times of concurrent processes. Anti-intuitive anomalies[23,3] are disturbing but do not keep us from conjecturing that they occur only infrequently. If we neglect anomalies, then we cannot expect change in execution time of any one instruction or any one routine or any one compiler to produce a decimal order of magnitude change in a sensibly weighted function of the above parameters. Given reasonable measurement tools and design of measurement experiments we conjecture that somewhere between 10 percent and 50 percent improvement in performance can be accomplished for most systems by changes in assignment and sequencing of resources. Although these percentages do not seem dramatic in their impact, the absolute number of dollars or number of computer hours which would become available is far from negligible.

In contrast with the heuristic probing and tuning of a given system, much greater impact is possible at the user interface with a computer system and by advances in models, particularly validated models of our computer systems. For example, we would guess that there are more than 10 attempts to run a program during its development before it runs once "correctly." For complex programs the ratio of number-of-correct-runs to number-of-runs can approach zero. Hence, if the user interface can be altered so as to increase the probability of a correct run, large benefits may result.

The effect of model development is a more sophisticated and qualitative issue. It is self evident that to the extent that we can predict behavior of even a subsystem through modeling, we can hope to isolate important parameters and the way they affect performance. In fact, only through modeling efforts can we generalize experimental results at one center to apply to many others. Furthermore, it has been recognized that simulation is the most widely used tool in evaluation of systems. If simulation depends upon precise imitation of a computer system, its development cost is generally prohibitive and it is fraught will all the unreliability associated with one-shot development. Effective simulation depends upon validated approximate models of systems and of user programs. Creation of such strong models is the most difficult of our tasks. However, the very process of validating or invalidating simplifying assumptions used in models can lead to new algorithms and improved models. Margolin, Parmelee and Schatzoff[39] very competently demonstrate this effect in their recent study of free-storage management algorithms.

In this section we have taken cognizance of the fact that there is no simple (or even complex) formula for computer performance. The reader's attention has been focussed on the last five in the list of factors affecting computer performance because they offer so much more return. The following sections review work in analytic modeling, measurement, and the user interface.

CRITIQUE OF ANALYTIC MODELING

Any system design, any measurement project or any resource allocation strategy is based on some conception of the environment in which it operates. That conception is a model. It is beneficial to have such models explicitly stated so that they can be explored, tested, criticized and revised. Even better, though not often achieved to the extent desired, is a formal analysis of the models.

Models and methods of analysis vary greatly. Our concern here is with probabalistic models of systems and processes and also with discrete graph models of programs. The goals of these analyses are both insight and quantitative results to influence the design of systems, resource allocation strategies and possibly the design of languages.

While most will argue that the goals of such analyses are inherently worthwhile and must be pursued, there is widespread dissatisfaction with the current state of the field. Basically, there are three major areas of dissatisfaction. First, the models are generally oversimplified in order to make them mathematically tractable. This obviously makes the results questionable and brings us to the second major failing which is that analytic results are often not validated by measurement or simulation. Moreover, in cases where system evaluation studies are carried out, the existing models do not seem powerful enough to provide a uniform basis for measurements. The third major criticism is that most of the literature on analytic modeling is a collection of analyses of specialized models. This points up the lack of very general powerful results which would allow analysis to become an engineering tool. As it is now, each new situation almost always requires a separate analysis by an expert.

While the above are substantial criticisms, this is not to say that analysis has not had its impact. We can cite, for example, the working set model of program behavior,[12] the work on stack algorithms,[41] studies of time-sharing and multiprogramming system resource allocation and analyses of I/O scheduling,[42,49] the work on data transmission systems and on networks[11,35,19] and the work on graph models of programs.[40,26,27,33,14,24,7,10,22]

Promising areas of research

Multiple resource models

Much analytic work has dealt with single resource models. The reason for this is clearly that most of the analytic tools which are available apply to single resource environments. The computer system analyst is typically not a mathematician developing new tools but is generally engaged in applying existing tools. Nevertheless, computer systems are multiple resource systems and we must learn to analyze such systems.

Some recent studies of multiple resource models of computer systems have been made using results by Gordon and Newell.[21] The general model considered by Gordon and Newell is one in which customers (or jobs) require only one resource at a time, but move from one resource to another. An example is illustrated in Figure 1 for three resources.

The nodes in this figure represent resources and the arcs represent possible transitions from one resource to another. When a customer has finished at resource i he moves to (requires) resource j next with probability P_{ij}. The arcs are labeled with these probabilities. The service time at each resource is assumed to be exponentially distributed. This is a closed system meaning that the number of customers in the system remains fixed. Gordon and Newell have found expressions for the equilibrium distribution of customers in service or queued at each resource. This allows one, for example, to calculate the utilization of the various resources.

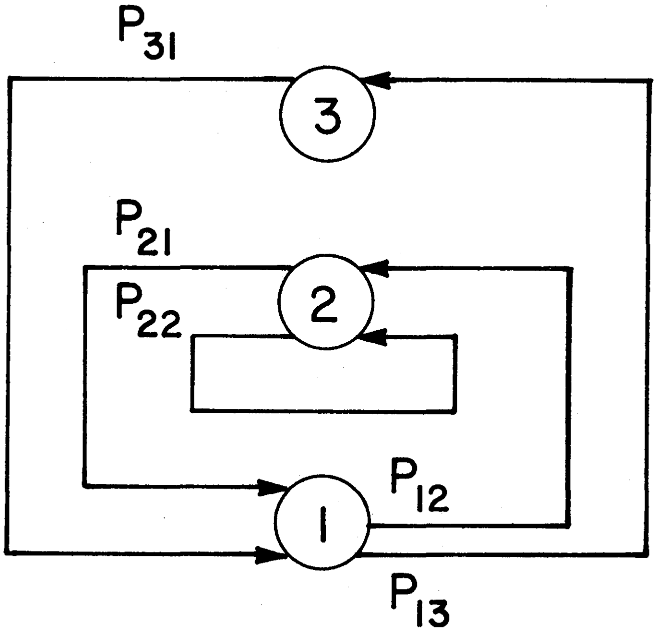


Figure 1—Example network of queues model

Moore[43] and Buzen[8] have applied this model to multiprogramming systems. Moore measured the MTS system to obtain the transition probabilities and mean service times of the resources and then used the model to estimate system parameters such as resource utilizations. The relatively close agreement to measured system parameters leads one to believe that the model can be used to predict the effect of some changes in system configuration. In using the model in this way, one must be careful that the proposed changes do not significantly affect the basic behavior of the customers. Buzen used the same model to gain insight into resource allocation in multiple resource models of computer systems. His studies include the investigation of buffering and the effects of paging algorithms. Both Moore and Buzen have used the model to try to give a meaningful formal definition to the term "bottleneck." It is of interest that they arrive at different definitions of a bottleneck. The reader is referred to the references for details.

While the studies mentioned above are clearly advances in the study of computer system models there are numerous open questions. For example, the model does not allow the representation of the simultaneous use of several resources such as memory and CPU. Also there is no means for representing the synchronization of events such as a process doing buffered I/O. Another limitation is that the customers in the system are assumed to have the same statistical behiavor, i.e., the transition probabilities and service time distribution are the same for all customers.

Bounds and approximations

Every evaluation technique makes use of approximations. These approximations may arise, for example: in estimating the system parameters, user and program behavior; or in simplifying the model of the system itself. There is clearly a tradeoff between types of approximations. By simplifying a model one might be able to handle more general types of user and program behavior. Much of the analytic work has been concerned with exact mathematical solutions to models which are themselves gross approximations.

An area which is beginning to be explored is that of approximate solutions to more general models. For example, Gaver has used the diffusion approximation for the analysis of heavily loaded resources in queueing studies.[20] The basic technique is to consider that the work arrival process is not the arrival of discrete customers requiring service but rather a work arrival flow. This work arrival flow is a continuous process with the same mean and variance as the original process. Another example of the use of approximations is the work by Kimbleton and Moore on the analysis of systems with a limiting resource.[34]

It is clear that the use of any approximation requires validation of the results. This may take the form of comparing results with measurements of an actual system, simulation, or obtaining bounds on the error in results. Bounds may also be applied in a different manner. Much has been written on the analysis of time-sharing scheduling algorithms and their effects on response times. Kleinrock, Muntz and Hsu[36] have reported on results which in effect demonstrate the bounds on response time characteristics for any CPU scheduling algorithm which does not make use of *a priori* knowledge of customers service times. The importance of the bounds is that one can see the limits of the variation in response characteristics that are possible by varying the scheduling algorithm and the extent to which these limits have been approached.

Program behavior

A major problem that must be dealt with in any evaluation effort concerned with computer systems is program behavior. Even when using approaches such as benchmarking or trace-driven modeling there is the problem of selection of programs which are in some sense representative of the total population of programs that will be run on the system.

Studies of memory management in particular have had to explicitly include models of program behavior. The early work in this area[2,12] stressed very general but powerful aspects of program behavior such as "locality" and "working set." More recent work deals with more explicit models of the generation of reference strings which assume more about program behavior but correspondingly allow for more detailed analysis.[13] It is hoped that these models will permit more detailed studies of multiprogramming and procedure sharing.

It is interesting to note that the bulk of this work has been directed toward finding models which can represent the universe of possible programs. More particular:y, the goals of this research have been to isolate parameters characterizing program behavior to which memory management is sensitive and to compare the effectiveness of various memory management strategies. This approach is in line with a common theme which runs through most of the work on resource allocation strategies in computer systems. That is, we see most allocation strategies attempting to work well over the total population of programs possibly utilizing measurements of recent past history of the process to predict the near future. Outside of work arising from graph models of parallel programs[5,6,51] very little has been done to utilize *a priori* information about a process. Many systems do make *a priori* distinctions between batch and interactive processes. It seems reasonable though that much more information may be available which would be useful in allocating resources. For example, it has been suggested that the time-slice and paging algorithm parameters be tailored to the process.[46] Use of *a priori* information assumes that the process is available for analysis prior to execution. This is a valid assumption for production jobs, system processes, and to some degree for all jobs at compile time. Since these processes consume a significant portion of the system resources, gains in efficiency in managing such processes might result in major gains in total efficiency. There are many open problems associated with this approach:

1. Is there a conflict with a program design goal of program modularity? How is information about separately compiled procedures to be combined?

2. Should processes be permitted to advise the system as to their resource needs? How does the system protect itself against false information?
3. How to manage resources effectively for processes which provide a priori information, and also for processes without associated *a priori* information?
4. What kind of *a priori* information is actually useful to management of a system: how costly is it to obtain and utilize effectively?
5. How predictable are the resource requirements of processes?

While this approach has received only some slight mention in the literature, it appears to be a fertile area for research.

Graph models of programs provide an abstraction of program structure governing flow of control and demand for resources.[51,10,18,22] They permit a representation fitting somewhere between the full detail of actual programs and parametric or stochastic representations of them. Most work using graph models has been concerned with *concurrent processing*. However, the graph model analyses explicitly reveal sets of independent tasks which become candidates for alternate sequencing *in sequential systems*.

Ideally, we search for models of systems and program behavior which provide principles guiding synthesis of configurations along with well founded resource management strategies. Measurement must validate effectiveness of such strategies. The diversity of computations further demands that measured parameters be provided to operating systems and users in order to permit adaptation to dynamic variations in system behavior and to unavoidable anomalies in systems and languages.

Studies during the latter half of the '60s showed how little attention had been given to measurability in the man-made universe of computer systems. The next section characterizes some of the problems in measurement.

MEASUREMENT OF INFORMATION PROCESSING SYSTEMS

Tools for measurement of computer systems must satisfy all of the following requirements: detection of prescribed events; recording of detected events; retrieval of accumulated records; data reduction; and display.

We comment on each in turn.

Detection

We start by rejecting the absurdity of observing all of the states of a system under observation since it would imply detecting the state of every input, every memory element and every output every time there was a change, along with the time at which the change occurred. Hence, any set of measurement tools must include means of selecting a subset of system states.

Hardware measurement tools provide a prescribed number of sensing probes which may be physically placed on selected register or buss points in a machine under observation. Measurement system registers along with programmed comparators and basic logical operations permit further filtering by allowing detection of a subset of the events sensed by the probes. Even with such filtering the rate of change of detected states may be excessive. If the response time of hardware measurement elements is insufficient, basic circuit changes would be required to make the measurement feasible. If bandwidth is insufficient, it is sometimes possible to introduce a sampling signal and thereby further reduce the number of detected events. *In the absence of interaction with software monitor programs, a hardware monitor is clearly limited in its utility.* To be convinced of this, one need only consider the kind of program status information change which is observable by probes only when it appears in the form of an operand during the course of computation. Hardware detection can have the virtue of introducing no artifact into the measured system and of being able to detect events whose states are not accessible to measurement programs. Sampled detection may be made more effective by allowing interference with the observed process. If a sampling signal enforces a proper interruption, observed data may be sequentially sensed by detection circuits. The recently reported "Neurotron" monitor[1] is the most interesting implemented hardware monitor, and its design shows the foresight of enabling interaction with software monitor programs.

Software measurement tools consist of programs which detect selected events by virtue of their insertion at state-change points in the sequential computational process.[17,47,31] This detection process introduces artifact in execution time, in space required for measurement program storage, and sometimes (e.g., synchronization with asynchronous cyclic processes) in qualitative side effects on existing computational processes. In a sampling mode, measurement programs can have their in-line artifact reduced by disturbing the flow of computation only at a sampling time. At a sampling time, measurement programs may be brought in to check as large a set of memory states as is needed and then control is returned to the observed system. In the absence of hardware support, a software monitor is limited to observation of those system states which have affected memory contents. In one case, careful analysis of measurement of PL/I functions of an IBM 360/91[50] revealed anomalies in recorded system states which can best be characterized as artifact introduced by OS/360 when it inserts code associated with I/O interrupts into the code being measured.

It has become clear that we are not faced with mutually exclusive alternatives of hardware detection tools or software detection tools. Rather how much of each; how they are integrated; and how they are made available to experimenters. A paper by Nemeth and Rovner[45] presents a pleasing example of the power of combined hardware and software in the hands of a user. They point out that facilities introduced for hardware debugging are often the kind useful in later program measurements.

Recording

If an event of interest has been detected, its occurrence must affect memory contents. Such action may be as simple as incrementing a counter or as complex as storing a lot of state information for later analysis. In the case of nondisturbing hardware measurements, external storage must be provided and the transfer rate must be able to keep up with the rate of change-of-state information observed by a set of probes and associated circuits. In the case of software measurements, sufficient memory space must either be provided to record all relevant state information, or else preprocessing reduction programs must be called in to reduce storage requirements.

Retrieval

In the construction of any large system, both a data gathering and retrieval system must be incorporated into the basic design. Failure to do so will limit the amount of instrumentation available later when efficiency questions arise. For example, in a large programming system which has transient program segments, data gathering is easily inserted into any program segment; however, unless a standard data storing program is available, the data gathered cannot be easily retrieved. The IBM PL/I F-level compiler is an example of a programming system broken into transient program segments. It fails to have a data storing program with adequate bandwidth to support meaningful measurement activity.

Data Reduction

The amount and kind of data reduction is determined by the goal of the measurement experiment and limitations of measurement tool capabilities in detection, recording and preparation for retrieval. For example, assume that we want to obtain a history of utilization of routines in order to decide which should be kept in primary storage and which should be kept in backup storage. Assume, further, that every time a routine is called: the name of the called routine, the time of day, and the name of the user is recorded. It would not be very meaningful to generate only a history showing the times at which each routine in the system was used by each user. Data reduction would be required to determine, for example, the total number of such uses, an ordering of routines by number of uses, a determination of the number of routines involved in, say 50 percent, of the uses and their names, etc.

Display

The goal of the data reduction process is defined by the specified form of display or feedback to the experimenter. If measurement is being made for feedback to an operating system for use in resource allocation, parameter values must be delivered to memory cells to be accessed by the operating system. If measurement is made for accounting purposes or, more generally, to provide the user with feedback about his quality of use and system response, results should be merged into user and system manager files. If measurement is made for operator control and management, simple alphanumeric displays are common. For experimental analysis of system behavior, CRT displays, graphs and computer printout are generally required.

Measurement Methodology

The complexity of computer systems dictates special care in the planning of measurement experiments. If the results of experiments are not reproducible, they are of little value. If any assumptions made are not recorded and validated, the results cannot be generalized and applied at another time or place. A large body of statistical theory is available providing methods for abstracting properties out of individual data points, but applicability must be carefully checked. We have little hope of adhering to principles if we do not have a measurement language to prescribe measurement experiments as sequences of commented operations which are appropriately integrated with observed data. The latter step needs wait upon creative development of measurement tools and their test in meaningful experiments. Measurement capability must be explicitly included during periods of system design and must be available for inclusion in user program design. Digital computer systems and programs are properly characterized as complexes of very elementary functions. Full systems or programs, therefore, generally require partition in order to manage the synthesis process. Each partition introduces possible measures of validity of output, of performance and of cost. Means for measurement should be checked at that point in design and a value judgment made if excessive artifact would be introduced by the measurement process.

If a system contains the structure and primitive operations satisfying the five requirements discussed in this section, it carries the tools for adaptability. We conjecture that much more than the 10 percent to 50 percent improvement alluded to in the Introduction becomes attainable—particularly when measurement tools can influence user behavior.

COMPUTER POWER AND USER INTERFACE

In the seventies some stronger effort must be directed toward increasing computer power by a reduction of the complexity of the user interface.

Operating systems and **H**igher **L**evel **L**anguages are tools designed to give the user control of the portion of a computer system he needs. With the exception of work done at SDC,[28,29] little reported effort has been devoted to the human engineering aspects of these tools. During the last decade, while hardware made a dramatic increase in power, the management tasks required of the operating system increased from trivial to highly complex. At the same time, the users were required to supply (in unnatural form like JCL) more of the parameters which would allow effective management decisions to be made by the operating system. These user-supplied parameters have increased the burden of complexity at the user interface—and reduced the amount of useful work a user can accomplish in a given period of time.

For example, much of the attraction of APL/360 is its simplification of the operating system interface along with the addition of immediate execution of its concise powerful primitives. A batch oriented FORTRAN user perceives this as a tremendous increase in his computer power. A more sophisticated user might see APL as a powerful desk calculator which provides immediate access to functions similar to functions he already commands, less accessibly, in other languages.

Another user interface problem exists at the level of higher level languages. As more advanced hardware becomes available to the user, he seeks to solve more complex problems. When a problem grows beyond a manageable point, the user segments the problem into pieces plus associated linkages. In doing so, however, he introduces a new set of communication problems; a change in one program which affects an interface can now wreak havoc in another "completed" portion of the problem solution. Higher level languages have been lax in the types of program interconnections (and interactions) allowed.

An example of the problems of creating a large programming system are reported by Belady and Lehman using data from the development of OS/360.[4] While this study concerned programs written in assembly language for the IBM 360, the properties which produce the error rates and modification ratios reported in their paper are characteristics of all large programming systems today.

Several techniques for improving the probabilities that a program can be made error free are available in the literature. One of the earliest is Dijkstra's "Notes on Structured Programming,"[16] and also "THE Programming System."[15] His system breaks the problem into shells of "pearls" or "primitive" operations. Each shell is built using the "primitives" of the next lower level. This system attempts to minimize interactions, forces the programmer to produce generalized functions which can be tested, and allows easy instrumentation because of the segregation of functions.

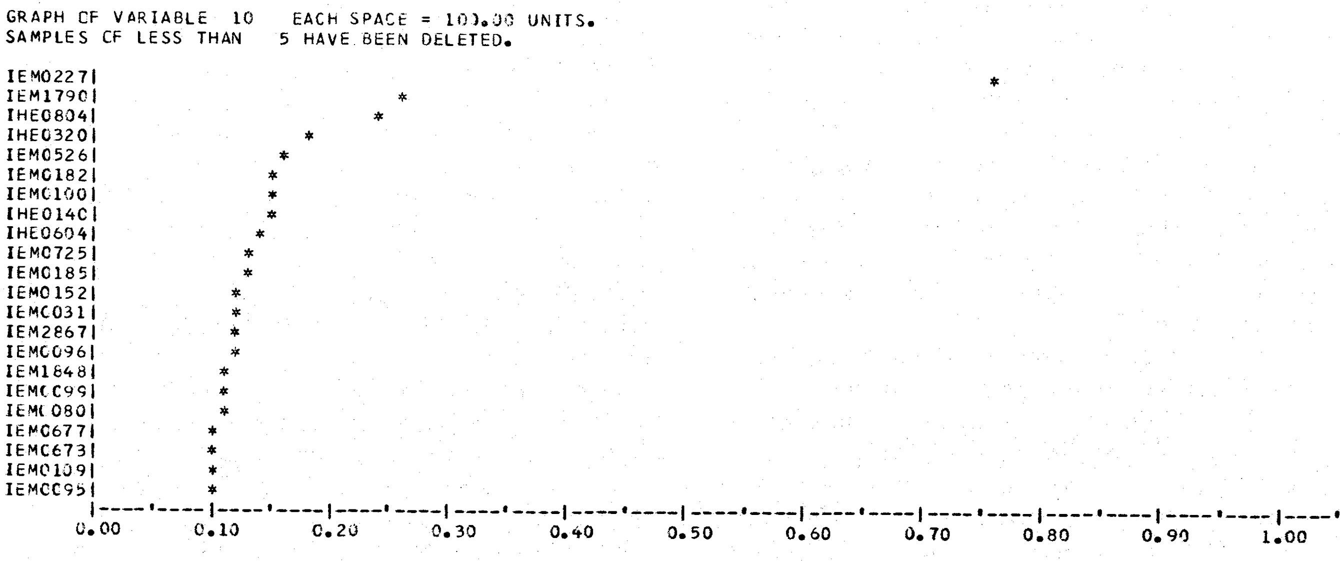
Some disadvantages of such a hierarchical scheme make its practical application difficult. Such a scheme increases initial development time because it forces developers to completely understand the structure of the system being built and to predefine the proper function for each hierarchy. Transitions between levels may be costly. Functions at the lowest level are the most general and, therefore, the most frequently used. Small inefficiencies in these functions, or in the method of traversing levels of the structural hierarchy, magnify costs dramatically and force the user away from centralized functions. This defeats the original purpose of the organization.

Another disadvantage of a hierarchical scheme is that while instrumentation of the system is easy, interpretation of the measurements is generally not. Measurement results could change drastically if the organization of the program were modified. Therefore, it is hard to tell how much of what goes on is due to the structural hierarchy and how much is due to the intrinsic properties of the program. Such knowledge points a way toward improvement.

	NUMBER OF ERROR OCCURRENCES	ERROR TYPE*	ERROR DESCRIPTION
COMPILE-TIME			
	263	IEM0227	NO FILE/STRING SPECIFIED. SYSIN/SYSPRINT HAS BEEN ASSUMED.
	87	IEM0182	TEXT BEGINNING yyyy SKIPPED IN OR FOLLOWING STMT NUMBER.
	74	IEM0725	STATEMENT NUMBER xxxx HAS BEEN DELETED DUE TO A SEVERE ERROR NOTED ELSEWHERE.
	63	IEM0152	TEXT BEGINNING yyyy IN STATEMENT NUMBER xxxx HAS BEEN DELETED.
	46	IEM1790	DATA CONVERSION WILL BE DONE BY SUBROUTINE CALLS.
	39	IEM0185	OPTION IN GET/PUT IS INVALID AND HAS BEEN DELETED.
	27	IEM0677	ILLEGAL PARENTHESIZED LIST IN STATEMENT NUMBER xxxx FOLLOWS AN IDENTIFIER WHICH IS NOT A FUNCTION OR ARRAY.
	27	IEM0109	TEXT BEGINNING yyyy IN OR FOLLOWING STATEMENT NUMBER xxxx HAS BEEN DELETED.
	25	IEM0096	SEMICOLON NOT FOUND WHEN EXPECTED IN STATEMENT xxxx. ONE HAS BEEN INSERTED.
	23	IEM0673	INVALID USE OF FUNCTION NAME ON LEFT HAND SIDE OF EQUAL SYMBOL OR IN REPLY, KEYTO OR STRING OPTION.
EXECUTION-TIME			
	14	IHE804	ADDRESSING INTERRUPT.
	12	IHE320	FIXED OVERFLOW.
	8	IHE140	FILE name—END OF FILE ENCOUNTERED.
	7	IHE604	ERROR IN CONVERSION FROM CHARACTER STRING TO ARITHMETIC.

* IBM, PL/I(F) *Programmers' Guide* (*Appendix K*), GC28-6594, January 1971.

Figure 2a—Most frequent PL/I errors



```
GRAPH CF VARIABLE  10    EACH SPACE = 10).00 UNITS.
SAMPLES CF LESS THAN   5 HAVE BEEN DELETED.
```

Figure 2b—Average persistence sorted by average persistence

	NUMBER OF ERROR OCCURRENCES	ERROR TYPE*	ERROR DESCRIPTION
COMPILE-TIME			
	143	SY16	IMPROPER ELEMENT(S)
	131	SYE5	ILLEGAL USE OF COLUMN 1 ON CARD
	89	CGOC	NO FILE SPECIFIED. SYSIN/SYSPRINT ASSUMED
	83	SY0B	MISSING SEMICOLON
	76	SM4E	ident HAS TOO MANY SUBSCRIPTS. SUBSCRIPT LIST DELETED
	55	SY3A	IMPROPER LABEL
	53	SY04	MISSING)
	48	SY06	MISSING COMMA
	46	SY09	MISSING :
	40	SM50	name NEVER DECLARED, OR AMBIGUOUSLY QUALIFIED (EXPRESSION REPLACED OR CALL DELETED)
EXECUTION-TIME			
	827	EX78	SUBSCRIPT number OF ident IS OUT OF BOUNDS
	239	EX83	FIXED POINT OVERFLOW
	211	EXBB	DELETED STATEMENT ENCOUNTERED
	189	EX7D	LENGTH OF SUBSTRING LENGTH OF STRING
	180	EX98	INCOMPATABLE OPTIONS ON OPEN
	169	EX7B	INDEX OF SUBSTRING LENGTH OF STRING
	141	EXB8	ARRAY ELEMENT HAS NOT BEEN INITILIZED. IT IS SET TO 0.
	90	EX9F	IMPLIED CONVERSION NOT IMPLEMENTED
	46	EX89	PROGRAM IS STOPPED. NO FINISH ON-UNIT AFTER ERROR
	45	EXB7	ident HAS NOT BEEN INITIALIZED.

* Conway, R. W. et al,. *User's Guide to PL/C, The Cornell Compiler for PL/I*, Release 6, Department of Computer Science, Cornell University, Ithaca, August 1, 1971.

Figure 3a—Most frequent PL/C errors

Present studies of forced program structure and program proofs of correctness may begin to provide models on which HLL designers may base their proposals. However, any major changes should be designed to improve the user's system so that each program submittal can be a learning experience. In this way a programming system can be called upon to point out unusual events; draw the programmer's attention toward possible errors; and yet, not produce volumes of output which would be costly to print and which a programmer would refuse to read.

Work at Cornell toward producing compilers which correct human failings rather than punish them has culminated in a highly functional, rapid compiling, and very permissive PL/I student-oriented compiler called PL/C.[44] This compiler does spelling correction of keywords, automatic insertion of missing punctuation, etc. In addition automatic collection of some statistics is done at execution time. For example, each label causes a count to be maintained of the number of times execution passed through that labeled statement. Compilers such as these increase computer power by reducing the complexity of the user interface.

Implementations of HLLs could further help a programmer by giving an optional cross reference listing showing locations where a variable is changed, could be changed, or just referenced. Items could be flagged if they were never referenced or set; only referenced; or only set. In the first two cases spelling correction might be applicable. Statements which use expensive library subroutines or other costly language features could be flagged. Measurement nodes could be easy to insert and operate. These should, in turn, produce meaningful data which relate directly to questions the programmer wanted to ask.

But such discussions have only beat around the bush itself. The real problem, the bush, is the higher level language. The real questions are: What features are error prone? What features of the language allow automatic validity checking of what is written? How can these properties be identified and measured? How can the knowledge of these things be used to reduce complexity of the user interface so that the user perceives an increase in his computer power? Which language constructs are seldom used, adding unnecessary complexity and unreliability?

Efforts to measure the human engineering aspects of computer language use[37] and to provide feedback

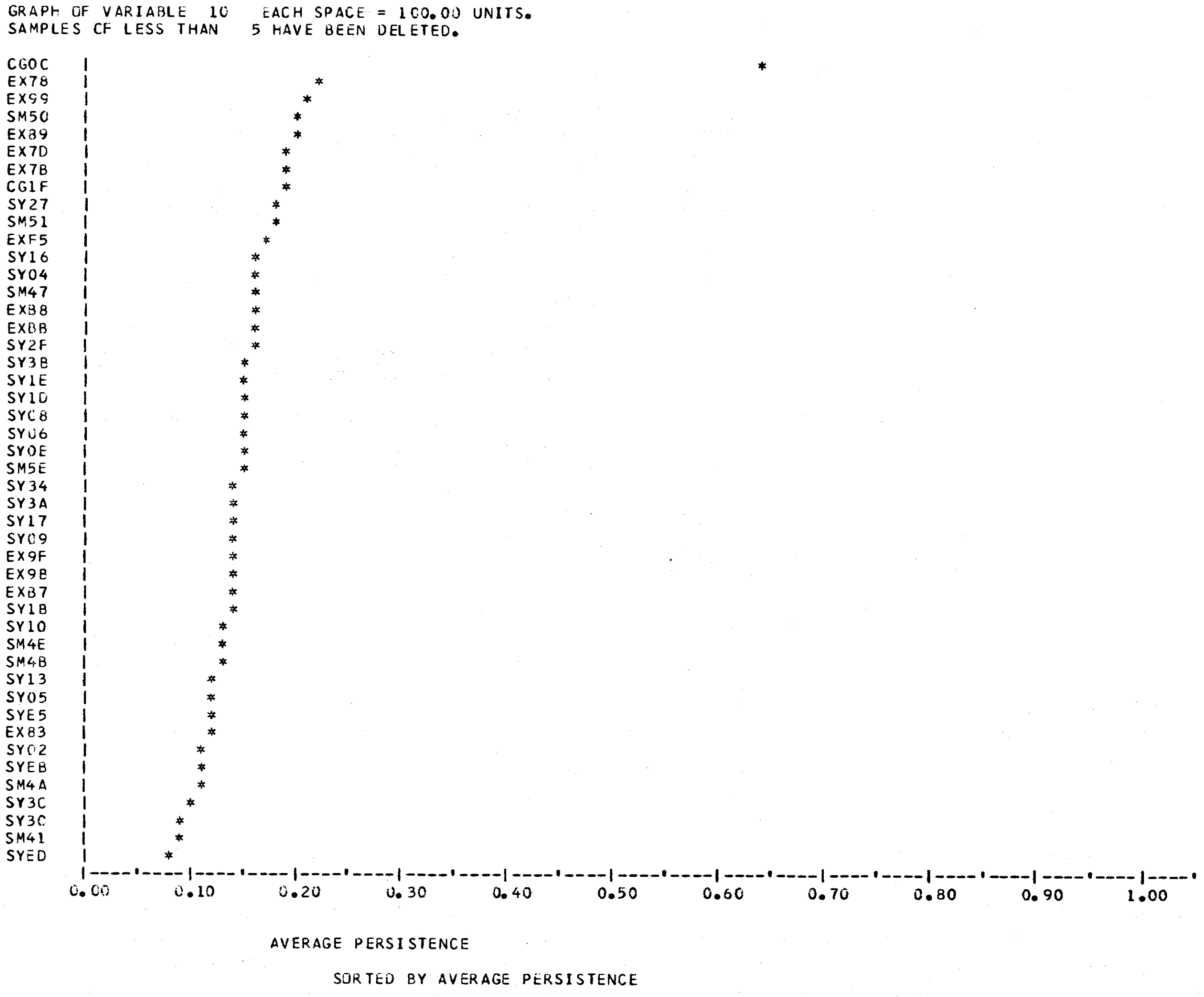


Figure 3b

into the design stages of higher level languages and the control and command languages of the operating system may provide major increases in computer power by:

- increasing the number of users who can bring problems to the machine
- decreasing the number of problem submissions necessary to bring a job to completion

Work is in progress at SDC,[28,29,48] moving toward a man-machine symbiosis. These little publicized approaches to measurement of human problem solving and computer languages have just begun to scratch the surface of this very important area. Work at UCLA has attempted to identify properties of PL/I which are prone to human error. As a first approximation, error rates and persistence curves of various errors identified in students' use of the IBM PL/I F-level compiler[30] is presented in Figure 2. Corresponding results for errors found by Cornell's PL/C compiler are presented in Figure 3. Figure 2a shows a table of the number of occurrences of the most frequent PL/I error types recorded during compilation and during execution times. Figure 2b displays the persistence of errors by PL/I type during the student runs. The vertical coordinate is the error type ordered by the magnitude of PERSISTENCE RATIO. The horizontal coordinate is the PERSISTENCE RATIO and was calculated as an average of (number of sequential trials during which the particular error persisted) divided by the total number of trials. If an error type did not occur in at least 5 problem assignments it was arbitrarily deleted to keep the displayed range of values reasonable. Figures 3a and 3b display the same properties for assignments using PL/C. A total of 128 problem assignments completed by 28 students are included in the statistics. Follow-up work is intended to lead more deeply into language design and hopefully into new techniques for automatically localizing errors in a program.

The basic technique for doing this is to allow the programmer to specify more information than the HLL processor needs to compile the program. An example would be identifiers of the form "CONSTANT" in a PL/I data attribute syntax. CONSTANTs as opposed to variables, would only be set by an initial attribute and would be illegal as a left-hand side of an assignment or as an argument of a pseudo-variable. In addition, the program could be considered as having this attribute. At several points in a program (e.g., block exit time) these constants would be checked to see if their value had changed. If any had, a warning would be printed; the correct value restored; and the program would continue. Such a new PL/I data type allows automatic checking for consistency to localize errors and yet is almost painless for a programmer to use. When a program is debugged, it is easy to turn off this kind of checking for the sake of more efficient performance. In a hardware environment like the MULTICS GE 645, these errors can be detected dynamically when illegal accesses occur.

Debugging tools should be designed into the language and taught as part of the language, because the majority of the time a programmer deals with a language, he is also dealing with an incorrect program. Subscript checking, trace information, validity checking at periodic intervals, time and count information, formatted displays of all program information and selective store and fetch recording are the kinds of things which should be available to the HLL programmer.

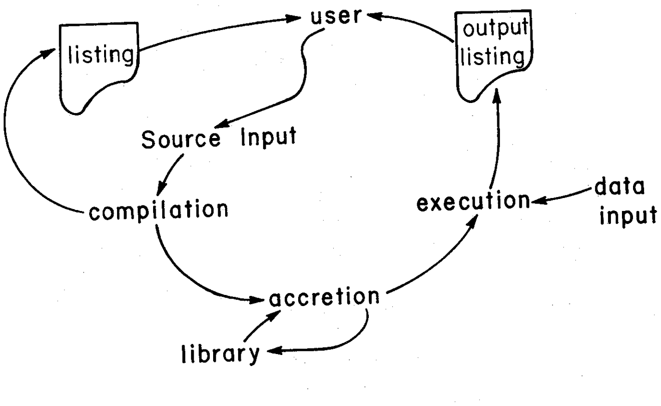


Figure 4a—Information flow in a standard HLL job

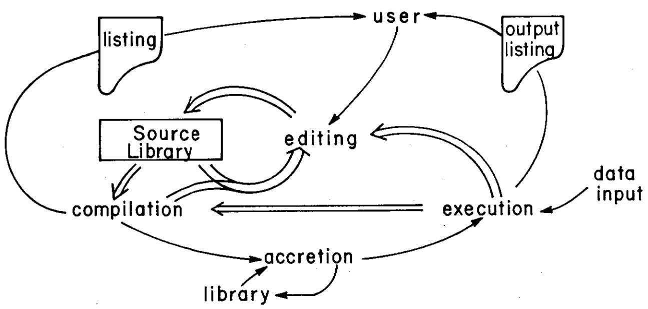


Figure 4b—Modified information flow to augment feedback in a HLL job

In addition, measurement tools should be immediately accessible to any user without burdening others, so that if questions of efficiency are raised they can be answered simply and quickly. Some of the measurement tools which seem important are: (1) flow charts or tables as optional output which would stress intermodule dependencies; (2) time and count control statements which could be output, reset and inactivated under program control, and would create output automatically if the program terminated abnormally or without expressly outputting the data gathered; (3) program size should be easily accessible by the program dynamically and summaries of available space should be available at program termination.

In order to increase the complexity of the programming problems which users can handle, languages must be allowed to accommodate personal ways of expressing concepts. To do this, at the very minimum, new data types should be available for the programmer to define as well as operators which use these data types. This begins to syntactically approach the Dijkstra concepts and to allow easier application of hierarchically structured programs. Hopefully these approaches will increase the user's computer power by making the development of his programs easier.

The programming system itself should be restructured so that more information is available to a HLL processor. Figure 4a shows the diagram of information flow in a usual batch oriented system. The source code is compiled; the resulting object code is passed to the accretion step where library or previously compiled programs are added to it; and the resulting load module is passed to the execution phase; finally, output from execution is passed back to the user. To provide more automated feedback, Figure 4b shows information flow in a system where statistical summaries of one execution are available to the next compilation. In addition, the accretion step spends much more of its time checking linkage conventions and the validity of argument-parameter choices. This programming system has an edit/compile time library which is designed to help make changes easy. For example, it keeps "COMMON" declarations uniform (read EXTERNAL if you are a PL/I programmer) and it also uses information from the compiler to point the user at syntax errors and information from the execution phase to point the user at semantic errors.

Such modifications can reduce errors and speed the development of programs by improving communication between what are now considered separate program steps. However, the most important changes, across all the proposed modifications, are those changes which will allow the programmer to receive only those pieces of information relevant to the level at which he is programming (i.e., making changes). This would provide dynamic help; help where the programming language acts as an extension of the users' mind to assist in problem solving and optimization.

It is important to view these changes which move toward dynamic assistance in terms of costs. Each change must cost something in execution time overhead. Some of the more powerful features like selective fetch and store monitoring must be expensive. However, if these features were found valuable, then modification to hardware might diminish costs dramatically. Integrating these techniques into HLLs must be inherently costly because implementation and testing of human interaction with these diagnostic features are difficult to execute in any controlled way—much work must rest on subjective evaluation of users' behavior. Integration of aids into HLL translators must be initially done without those very aids which are deemed necessary to help programmers modify programs. Therefore, any change is fraught with risks caused by lack of checks in current systems. Obviously bootstrapping is called for and we can expect many passes before achieving effective tools.

The development of richer higher level languages on the one hand, and the development of debugging services and error correcting compilers on the other, exert forces in the direction of increasing performance at the user interface. With appropriate use of models and measurement much more improvement may be obtained.

SUMMARY

Computer systems are different from other systems by virtue of the dynamic fashion in which our comprehension of their behavior may be built into their operation. If validated models are developed, they may then be built into the system itself to serve adaptive resource allocation algorithms. If measurement tools are effectively integrated, they may be made available to the user to improve the quality of his use of programming languages. If the user is, in fact, a team developing programming systems, the modeling and measurement facilities may serve to make much more complex programs possible because a model of programs being built is, itself, generally too complex for a group of unaided humans to manage in an error free way. In the above paper we have sought to open up these questions.

We have not had much experience with effective modeling and measurement. There is an immense amount of data to be observed in a computer system. Cost-effectiveness of performance measurement must be considered. As one of our reviewers put it, "This reviewer has seen some measurement studies lead to system improvements which will pay off sometime in 2018." Hopefully the 1970s will see more effective modeling and measurement introduced into the design process and selectively carried into developed systems to he`p both interna! process management and the enrichment of external usↄ through the user interface.

REFERENCES

1 R A ASCHENBRENNER L AMIOT
N K NATARAJAN
The neurotron monitor system
AFIPS Conference Proceedings Vol 39 pp 31-37 1971 Fall Joint Computer Conference Las Vegas Nevada November 1971

2 L A BELADY
A study of replacement algorithms for a virtual storage computer
IBM Systems Journal Vol 5 No 2 pp 78-101 1966

3 L A BELADY R A NELSON G S SHEDLER
An anomaly in the space-tim? characteristics of certain programs running in paging machines
Communications of the ACM Vol 12 No 6 pp 349-353 June 1969

4 L A BELADY M M LEHMAN
Programming system dynamics or the meta-dynamics of systems in maintenance and grow h
IBM Report No RC 3546 September 1971

5 D P BOVET G ESTRIN
A dynamic memory allocation in computer systems
IEEE Transactions on Computers Vol C-19 No 5
pp 403-411 May 1970

6 D BOVET G ESTRIN
On static memory allocation in computer systems
IEEE Transactions on Computers Vol C-19 No 6
pp 492-503 June 1970

7 T H BREDT
Analysis of paral'el systems
IEEE Transactions on Computers Vol C-20 No 11
pp 1403-1407 November 1971

8 J P BUZEN
Queueing network models of multiprogramming
PhD Dissertation Harvard University Cambridge Mass
August 1971

9 V G CERF
Measurement of recursive processes
Computer Science Department Technical Report No
UCLA-ENG-70-43 University of California Los Angeles
May 1970

10 V G CERF K GOSTELOW S VOLANSKY
E FERNANDEZ
Formal properties of a graph model of computation
Computer Science Department Technical Report
No UCLA-ENG-7178 December 1971

11 W W CHU
A study of asynchronous time division multiplexing for time sharing computers
AFIPS Conference Proceedings Vol 39 pp 669-678 1971
Fall Joint Computer Conference Las Vegas Nevada
November 1971

12 P DENNING
The working set model for program behavior
Communications of the ACM Vol 5 No 11 pp 323-333
May 1968

13 P DENNING S C SCHWARTZ
Properties of the working set model
Proceedings of the Third Symposium on Operating Systems
Principles pp 130-140 Stanford University October 1971

14 J B DENNIS
Programming generality, paralle'ism and computer architecture
Proceedings of the IFIP Congress 68 Booklet C Software 2
pp C1-C7 North Holland Publishing Co Edinburgh England
August 1968

15 E W DIJKSTRA
The structure of the THE multiprogramming system
Communications of the ACM Vol 11 No 5 pp 341-346
May 1968

16 E W DIJKSTRA
Notes on structured programming
Report No EWD249 Technische Hogeschool Eindhoven
Spring 1969

17 G ESTRIN D HOPKINS B COGGAN
S D CROCKER
SNUPER COMPUTER—Instrumentation automation
AFIPS Conference Proceedings Vol 30 pp 645-656 1967
Spring Joint Computer Conference Atlantic City New
Jersey April 1967

18 E B FERNANDEZ
Restructuring and scheduling of parallel computations
Presented at the Fifth Asilomar Conference on Circuits and
Systems Pacific Grove California November 8-9 1971 To be
published in the proceedings of that conference

19 H FRANK I T FRISCH W CHOU
Topological considerations in the ARPA computer network
AFIPS Conference Proceedings Vol 3 pp 581-587 1970
Spring Joint Computer Conference Atlantic City New
Jersey May 1970

20 D P GAVER
Diffusion approximations and models for certain congestion problems
Journal of Applied Probability Vol 5 pp 607-623 1968

21 W J GORDON G F NEWELL
Closed queueing systems with exponential servers
ORSA Journal Vol 15 No 2 pp 254-265 March 1967

22 K GOSTELOW
Flow of control, resource allocation, and the proper termination of programs
Computer Science Department Technical Report No
UCLA-ENG-7179 University of California Los Angeles
California December 1971

23 R L GRAHAM
Bounds for certain multiprocessing anomalies
The Bell System Technical Journal pp 1563-1581 November
1966

24 A N HABERMANN
Prevention of system deadlocks
Communications of the ACM Vol 12 No 7 pp 373-377 July
1969

25 L E HART G J LIPOVICH
Choosing a system stethoscope
Computer Decisions Vol 3 No 11 pp 20-23 November 1971

26 A W HOLT H SAINT R M SHAPIRO
S WARSHALL
Final report for the information system theory project
Rome Air Development Center by Applied Data Research
Inc Contract No AF30(602)-4211 1968

27 A W HOLT F COMMONER
Events and conditions
Parts 1-3 Applied Data Research Inc 450 Seventh Avenue
New York New York 10001 1969

28 A HORMANN A LEAL D CRANDELL
User Adaptive Language (UAL): A step towards man-machine synergism
SDC TM-4539 June 1969

29 A HORMANN S KAUFMANN-DIAMOND
C CINTO
Problem solving and learning by man-machine teams
SDC TM-4771 July 1971

30 *PL/I(F) compiler program, logic manual*
GY28-6800 Fifth Edition IBM 1966, 67, 68, 69 October 1969

31 D H H IGNALLS
FETE—A FORTRAN Execution Time Estimator
Computer Science Department Report No STAN-CS-71-204
Stanford University February 1971

32 R R JOHNSON
Needed: A measure for measure
Datamation pp 22-30 December 15 1971

33 R M KARP R E MILLER
Parallel program schemata
Journal of Computer and System Sciences Vol 3 No 4
pp 147-195 May 1969

34 S R KIMBLETON C G MOORE
A limited resource approach to system performance evaluation
Technical Report No 71-2 Department of Industrial
Engineering University of Michigan June 1971

35 L KLEINROCK
Analytic and simulation methods in computer network design
AFIPS Conference Proceedings Vol 36 pp 569-579 1970 Spring Joint Computer Conference Atlantic City New Jersey May 1970

36 L KLEINROCK R R MUNTZ J HSU
Tight bounds on the average response time for time-shared computer systems
Proceedings of the IFIP Congress 71 TA-2 pp 50-58 Ljubljana Yugoslavia August 1971

37 D E KNUTH
An empirical study of FORTRAN programs
Computer Science Department Report No CS-186 Stanford University 1971

38 H C LUCAS JR
Performance evaluation and monitoring
ACM Computing Surveys Vol 3 No 3 pp 79-91 September 1971

39 B H MARGOLIN P P PARMELEE M CHATZOFF
Analysis of free-storage algorithms
IBM Systems Journal Vol 10 No 4 pp 283-304 1971

40 D F MARTIN
The automatic assignment and sequencing of computations on parallel processor systems
PhD Dissertation and Computer Science Department Technical Report No UCLA-ENG-66-4 University of California Los Angeles 1966

41 R MATTSON J GECSEI D SLUTZ I TRAIGER
Evaluation techniques for storage hierarchies
IBM Systems Journal Vol 9 No 2 pp 78-117 1970

42 J M McKINNEY
A survey of analytical time-sharing models
Computing Surveys Vol 1 No 2 pp 105-116 June 1969

43 C G MOORE
Network models for large-scale time-sharing systems
PhD Dissertation University of Michigan April 1971

44 H MORGAN R A WAGNER
PL/C—The design of a high performance compiler for PL/I
AFIPS Conference Proceedings Vol 38 pp 503-510 1971 Spring Joint Computer Conference Atlantic City New Jersey May 1971

45 A G NEMETH P D ROVNER
User program measurement in a timed-shared environment
Communication of the ACM Vol 14 No 10 pp 661-666 October 1971

46 J RODRIGUEZ-ROSELL
Experimental data on how program behavior affects the choice of schedular parameters
Proceedings of the Third Symposium on Operating Systems Principles pp 156-163 Stanford University October 1971

47 E C RUSSELL G ESTRIN
Measurement based automatic analysis of FORTRAN programs
AFIPS Conference Proceedings Vol 34 pp 723-732 1969 Spring Joint Computer Conference Boston Massachusetts May 1969

48 H SACKMAN
Man-computer problem solving
Auerbach Publishers Inc 1970

49 T J TEOREY T B PINKERTON
A comparative analysis of disk scheduling policies
Proceedings of the Third Symposium on Operatin; Systems Principles pp 114-121 Stanford University October 1971

50 R C UZGALIS J ALLEN
360 model 91 execution times of selected PL/I statements
Modeling and Measurement Note No 7A September 1971
360 assembly language source code for selected PL/I statements with model 91 execution times
Modeling and Measurement Note No 7B September 1971 Computer Science Department University of California Los Angeles

51 S A VOLANSKY
Graph model analysis and implementation of computational sequences
PhD Dissertation and Computer Science Department Technical Report No UCLA-ENG-70-48 University of C.lifornia Los Angeles 1970

Experiments in page activity determination

by JOHN G. WILLIAMS

RCA Laboratories
Princeton, New Jersey

INTRODUCTION

In a memory hierarchy various storage devices are structured into various levels. By convention, information at storage level i may be accessed and stored in less time than information at storage level $i+1$. Since a smaller access time generally implies a greater cost per bit of storage, storage level i will generally be smaller than storage level $i+1$. An example of such an arrangement with four levels is a system with core, drum, disc and tape memory devices.

A storage level is called *directly addressable* if the information stored at that level may be utilized by the processor directly, without first being moved to another level. For example, in the core, drum, disc, tape hierarchy only the core memory is directly addressable, since information at the other levels must be moved into the core before it can be used by the processor. If a memory hierarchy contains two or more directly addressable storage levels, then these levels may themselves form a directly addressable hierarchy.

Recently a device called "large-core storage" has come into use. This device utilizes the familiar principles of core operation, only it is designed to have a greater access and cycle time, and consequently it is less expensive per bit. This allows a larger memory to be built for a given cost, so that large-core storage may be considered as a replacement for drum memory in a storage hierarchy. A four-level hierarchy might be formed using core, large-core, disc and tape. In this case storage levels 1 and 2 would form a directly addressable hierarchy.

Let us consider this directly addressable hierarchy in more detail. Information in the large-core store can be accessed in one of two ways. It can be moved from large-core to core (as if the large-core were not directly addressable), and then accessed from the core. Alternatively, the information can be accessed directly in the large core. Assuming that the information is moved in fixed-size units called pages, the following question presents itself: should a page be moved from large-core to core when it is needed, or should it be addressed in the large-core directly?

Algorithms which attempt to resolve this question will be considered below. Such an algorithm will be called a *page promotion procedure*, since its function is to promote a page to a more favorable position in the memory hierarchy. Although the following discussion will be in terms of a core, large-core hierarchy, some of the results might be applied to any directly addressable hierarchy with two or more levels.

In a way this question resolves itself quite simply. There is a certain overhead associated with moving a page from large-core to core. However, once a page has been moved it may be addressed by the processor in a shorter time. Thus a page should be moved if it is addressed often enough that the shorter access time of the core more than defrays the overhead costs associated with moving the page. In other words, those pages which are addressed more often (termed the more

TABLE I—Properties of the four programs studied

PROGRAM		NUMBER OF PAGES (J)	NUMBER OF TIMES ADDRESSED (N)	DESCRIPTION OF PROGRAM
1	INSTRUCTIONS	35	833,907	FORTRAN IV COMPILER
	DATA	45	397,075	
2	INSTRUCTIONS	75	1,877,187	SNOBOL
	DATA	105	1,140,077	
3	INSTRUCTIONS	31	209,973	COBOL COMPILER
	DATA	48	137,530	
4	INSTRUCTIONS	27	973,786	ASSEMBLER
	DATA	46	641,931	

active pages) should be moved. The problem is to determine which pages are the more active.

Algorithms which detect the high activity pages may be divided into two types. Algorithms of the first type simply assume that page activity is known prior to processing. This is a reasonable assumption for programs of known characteristics. For example, it may be known that certain pages in a compiler or in an operating system are more active than are others. This approach has been implemented by Vareha, Rutledge and Gold[3] at Carnegie-Mellon University, and by Freeman[1,2] at the Triangle Universities Computation Center.

Algorithms of the second type depend upon the statistical properties of page activity, although they assume no *a priori* knowledge of the activity of any particular page.* Such an algorithm must be used when a program is not run often enough to become "known" in the sense of the first approach discussed above. Even when a program is known, algorithms of this second type may still be effective and may be simpler to implement. The page promotion procedures to be studied below are algorithms of this second type.

In the next section we will introduce some experimental data concerning the page activity of several programs. The two sections following this are devoted to an analysis of page promotion procedures. In these two sections, the experimental data will be used to test the effectiveness of the procedures.

EXPERIMENTAL DATA

In the previous discussion it was tacitly assumed that certain pages of a program will be addressed more often than will others. If this is not so, then the page promotion procedure is pointless. We will now present some experimental evidence which shows, at least for several programs, that some pages are indeed addressed more often than are others.

We consider the behavior of four programs, each of which addresses memory for both instructions and data. Let J be the number of pages addressed, and let N_j be the number of times page j is addressed ($j=1, \ldots J$), where a page is addressed either to access information or to store it away. For each program there are two sets of J and $N_1, \ldots N_J$ defined, one for instruction pages and one for data pages. Let the pages be numbered so that $N_j \leq N_{j+1}$ ($j=1, \ldots, J-1$), and define $N=N_1+N_2+\cdots+N_J$. Some of the properties of these four programs are summarized in Table I.

For a given program consider the x percent of the pages (either instructions or data) which are the most active. We wish to determine the percent of the total activity which these pages account for. Since $N_j \leq N_{j+1}$, this can be obtained as:

$$\text{Percent of Pages} = 100(k/J),$$

$$\text{Percent of Activity} = 100 \sum_{j=J-k+1}^{J} N_j \Big/ N,$$

where $k=1, 2, \ldots J$.

Figure 1 shows this result for the instruction and data pages of the four programs studied. Note that some pages are much more active than are others. Typically, the 50 percent of the pages which are the most active account for about 95 percent of all the activity. Thus, at least for these four programs, it is apparent that a page promotion procedure could be effective, if one can be devised. We will now discuss one such algorithm.

A SEQUENCE-INDEPENDENT ALGORITHM

For a given program, suppose that all of the pages are in large-core storage at the start of processing. Each time a page is addressed, assume that there is a fixed probability P that the page will be moved from large-core to core. Once a page has been moved, assume that it remains in core storage until processing is finished.

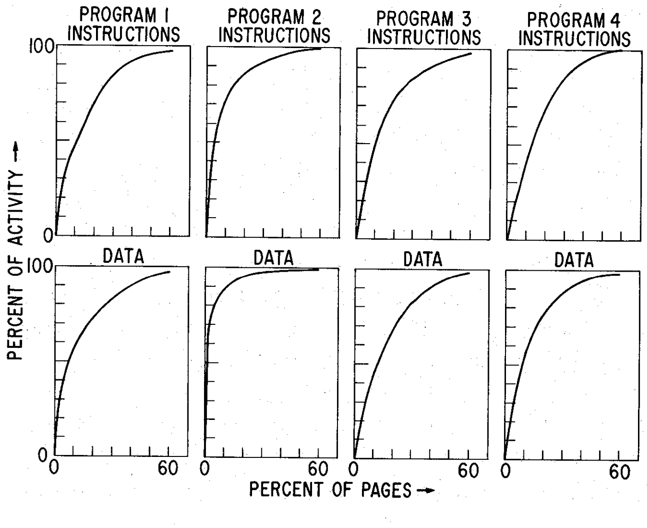


Figure 1—Page activity of the most active pages

* It might be noted that this classification of algorithms has an analogue in other resource scheduling problems. For example, in queueing theory the shortest-remaining-processing-time discipline requires an *a priori* knowledge of the processing time of each job; while the round-robin discipline achieves its effect due to the statistical properties of the distribution of processing times, with no *a priori* knowledge of the processing time of any particular job.

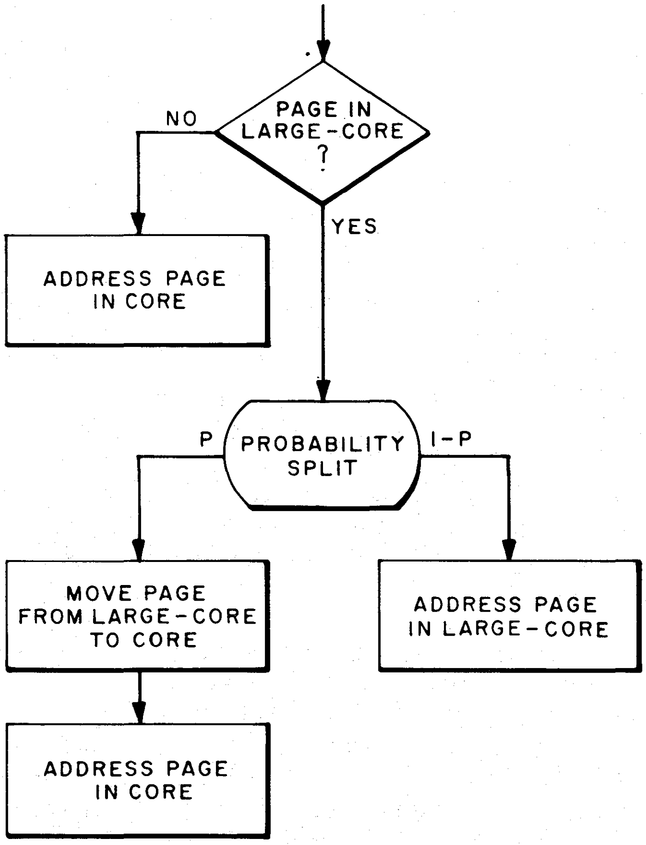


Figure 2—Sequence-independent algorithm

This, in brief, is the sequence-independent page promotion procedure which will be studied below. A flow chart of this algorithm is shown in Figure 2.

This algorithm is called "sequence-independent" because it acts on each page in a way which is independent of its action on all the other pages, so it does not depend upon the particular sequence in which the pages are addressed. This aspect will be considered in more detail when we discuss "sequence-dependent" algorithms below.

From a hardware standpoint, the implementation of the sequence-independent algorithm should be relatively straightforward. Suppose that we have a page-oriented machine which supports a translation between virtual and real page addresses. Then whenever a byte of any page is addressed, this translation function must be performed. At that time, a real-address limit comparison may be made to determine if the page is in large-core storage or core storage. If the page is in large-core, the output of a random number generator may be obtained from a special register. Assume that this output has a uniform distribution between 0 and 1. Then if the random number is between 0 and P, the page may be marked to be moved from large-core to core. This may be done by generating a page-fault interrupt and temporarily suspending the program using the page, just as if the page were on a drum or disc. While this is taking place, the random number generator may place a new number (independent of the previous one) into the special register. It should also be possible to place the probability P under program control, perhaps by keeping P in another special register.

It is clear that this algorithm will tend to detect the higher activity pages in the following sense. Given that a page is addressed more often, it stands a greater chance of *eventually* being moved from large-core to core. However, when it is moved, most of the addressing of the page may have already taken place. While it is clear that the algorithm will tend to tell which *were* the most active pages, it is not clear that it will indicate this while they still *are*. In other words, the algorithm itself must use the activity of the pages to select those pages with the higher activity. Thus there is a danger that the algorithm will "use up" the activity in the selection process, leaving little or none remaining after the page is selected. For any particular program, this question can be answered by the analysis to follow.

For a given program, suppose that page j is addressed N_j times, each time using the page promotion procedure described in Figure 2. Define:

λ_j The expected number of times page j is addressed in the large-core storage.

ϕ_j The probability that page j will not be moved from large-core storage to core storage.

It will be shown in Appendix I that:

$$\lambda_j = [(1-P)(1-(1-P)^{N_j})]/P,$$

$$\phi_j = (1-P)^{N_j}.$$

For the arbitrary program in question, we further define:

PA The expected percent of the N times that memory is addressed which will be directed to core storage.

PP The expected percent of the pages which will be moved from large-core storage to core storage.

Since the algorithm acts upon each page in an independent fashion, it follows that:

$$PA = 100\left[N - \sum_{j=1}^{J} \lambda_j\right] \Big/ N,$$

$$PP = 100\left[J - \sum_{j=1}^{J} \phi_j\right] \Big/ J.$$

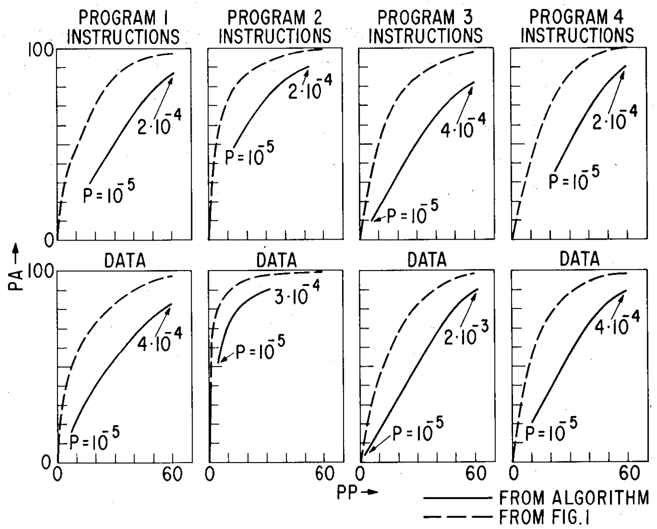


Figure 3—Effect of the sequence-independent algorithm

At the start of processing all the pages of a program are assumed to be in large-core storage. The page promotion procedure will cause a certain number of these pages to be moved from large-core to core. The expected percent of the pages which will be moved is *PP*. Because these pages are moved, a certain percent of the *N* times that memory is addressed will be directed to core storage. The expected value of this percentage is *PA*. We are willing to move a certain percentage of the pages from large-core to core, i.e., we are willing to accept a certain value of *PP*. For this value, we wish *PA* to be as large as possible. The value of *PA* will be increased by the extent to which the more active pages are selected soon enough.

Suppose that we are willing to move x percent of the pages from large-core to core. Then any random selection of x percent of the pages would account for an expected x percent of the activity. Thus a page promotion procedure in which $PA = PP$ is really useless, while one in which $PA < PP$ is less than useless.

Suppose that we are willing to move x percent of the pages from large-core to core, and that we know the activity of each page prior to processing. Then we would simply move the x percent of the pages which are the most active. For the four programs discussed in the previous section, Figure 1 shows the percent of the total addressing activity which would be directed to the core memory. For these programs, Figure 1 also provides an upper bound on the value of *PA* obtainable from the page promotion procedure for any given *PP*.

Applying the above equations for *PA* and *PP* to the data for the four programs discussed in the previous section, we obtain Figure 3. Since the performance of the page promotion procedure can be varied by changing the value of the probability *P*, various points on the curves are obtained by using various values of this parameter. The data from Figure 1 is also included in Figure 3, in order to compare the best possible performance of the procedure with the expected performance.

Various conclusions may be drawn from Figure 3. As an example, if we are willing to move 50 percent of the pages from large-core to core, then:

(1) By random selection of the pages prior to any processing, we could expect 50 percent of the addressing to be directed to core storage.
(2) Using the page promotion procedure, we could expect about 80 percent of the addressing to be directed to core storage.
(3) If we had perfect knowledge of the page activity prior to processing, we could expect about 95 percent of the addressing to be directed to core storage.

In this sense, the page promotion procedure is about $(80-50)/(95-50) = 2/3$ as effective as it possibly can be (when operating in the range of $PP = 50$ percent, on the four program studied). For smaller values of *PP*, similar conclusions could be drawn. However $PP = 50$ percent is an interesting value, since it reduces the page traffic between large-core and core by one half, and since at this value *PA* is about 80 percent, which is large enough to be of interest.

We note that *PA* and *PP* are expected values, so there is no reason to believe that the page promotion procedure will produce these values on any particular run of any particular program. However, it is assumed that a system using this procedure would process a large number of jobs. Thus, on the average, the specified values of *PA* and *PP* could be achieved.

A SEQUENCE-DEPENDENT ALGORITHM

Suppose that we modify the algorithm of the previous section in the following way. Let us replace the constant probability *P* by a probability P_k which is a function of k, the number of pages already moved from large-core to core. Such a modification would not be difficult to implement, since we have already assumed that *P* can be placed under program control. Such an algorithm will be termed *sequence dependent*, since the algorithm depends not only upon the number of times each of the various pages is addressed (the $N_1, N_2, \ldots N_J$), but also upon the sequence in which the pages are addressed as well.

In particular, we shall be interested in the sequence-dependent algorithm in which:

$$P_k = \begin{bmatrix} P & k=0, 1, \ldots, K-1 \\ 0 & k=K \end{bmatrix} \qquad (1)$$

This algorithm operates just like the sequence-independent algorithm until K pages have been moved from large-core to core. At this point the procedure simply shuts down, and no more pages may be moved. We are interested in this algorithm because it guarantees that no more than K pages will be moved from large-core to core. With the sequence-independent algorithm there is always the possibility that most or even all of the J pages will be moved. Let us now consider the extent to which this guaranty degrades the performance of the original sequence-independent algorithm.

In studying the sequence-independent algorithm it was only necessary to know the J integers N_1, N_2, $\ldots$, N_J, which are the counts of how many times the various pages are addressed. In order to study the sequence-dependent algorithm, it might appear that we must know the sequence in which the pages are addressed. This sequence could be characterized by a sequence of N integers, where integer n ($n=1, \ldots, N$) is j ($1 \leq j \leq J$), indicating that page j is the object of memory reference n. Using this approach, we could take the characterization of any particular sequence and find expressions for PA and PP.

An alternative approach would be to proceed as follows. For a program with a given N_1, N_2, $\ldots$, N_J, there are

$$N!/(N_1!\,N_2! \cdots N_J!)$$

different sequences in which the pages may be addressed. We shall establish that one of these sequences (called the *MIN* sequence) causes PA to take on its smallest value. If we can show that the sequence-dependent algorithm will still perform well under this sequence, then we can be assured that the algorithm will perform at least as well under all the other possible sequences.

There are several advantages to this second approach. One advantage, of course, is that less data is involved in the analysis of any particular program. To characterize a sequence we need N (the number of times addressed) data points, while with this second approach we will need only J (the number of pages).

There is a more fundamental reason for preferring this second approach. If the sequence-dependent algorithm were analyzed for one specific sequence, then we would have to consider the possibility that a slight change in this sequence might affect the performance of the algorithm. By showing that the algorithm will work across all possible sequences, we are depending upon a program property which is somehow less "fragile" than the addressing sequence, namely the addressing counts $N_1, N_2, \ldots, N_J$.

Let us now discuss the *MIN* sequence. Suppose that we have a sequence-dependent algorithm of the type specified by Equation (1), with given values for P and K. Consider a sequence of the following type: each time a page is to be addressed, the page is selected so that no other page has fewer remaining times to be addressed. Intuitively, with this sequence there is the greatest probability that the algorithm will move those pages with the least remaining activity. Since by convention $N_j \leq N_{j+1}$, one such sequence of this type is the one in which all of the addressing of page j is performed prior to any of the addressing of page $j+1$ ($j=1, 2, \ldots, J-1$). This particular sequence will be called the *MIN* sequence. In Appendix II we will show that of all the possible sequences with a given $N_1, N_2, \ldots, N_J$, the *MIN* sequence does indeed minimize PA.

We now wish to find expressions for:

PA^* The value of PA (the percent of the addressing activity directed to core storage) when the sequence-dependent algorithm is used with the *MIN* sequence.

PP^* The value of PP (the percent of the pages moved from large-core to core) when the sequence-dependent algorithm is used with the *MIN* sequence.

In the *MIN* sequence all the addressing of page j is performed at one time—just after the addressing of page $j-1$ has been completed and just before the addressing of page $j+1$ begins. We therefore define the following for the *MIN* sequence:

$R_{j,k}$ The probability that core storage contains exactly k pages just prior to the time that the addressing of page j begins.

$\lambda_{j,k}$ The expected number of times that page j is addressed in the large-core storage, given that core storage contains exactly k pages just prior to the time that the addressing of page j begins.

$\phi_{j,k}$ The probability that page j will not be moved from large-core storage to core storage, given that core storage contains exactly k pages just prior to the time that the addressing of page j begins.

It then follows that:

$$PA^* = 100\left[N - \sum_{j=1}^{J} \sum_{k=0}^{j-1} R_{j,k}\lambda_{j,k}\right] \bigg/ N, \qquad (2)$$

$$PP^* = 100\left[J - \sum_{j=1}^{J} \sum_{k=0}^{j-1} R_{j,k}\phi_{j,k}\right] \bigg/ J. \qquad (3)$$

From Appendix I we know that:

$$\lambda_{j,k} = [(1-P_k)(1-(1-P_k)^{N_j})]/P_k,$$

$$\phi_{j,k} = (1-P_k)^{N_j}.$$

The probabilities $R_{j,k}$ may be computed by an algorithm to be developed in Appendix III. Using this algorithm and the above four equations, PA^* and PP^* may be determined.

Although this method may be used to compute PA^* and PP^* for any sequence-dependent algorithm, we will limit our interest to the algorithm specified by Equation (1). Thus each time a page in large-core storage is addressed, there is a probability P that the page will be moved to core storage. When K pages have been moved, the promotion procedure will be terminated and no more page movement will take place.

To use this procedure the parameters K and P must be set. To set K let us suppose that there is a certain maximum percentage of the pages which we wish to move from large-core to core. Let this percentage be called PP_{MAX}. Then:

$$K = [PP_{MAX}/100] \cdot J. \qquad (4)$$

Consider now the problem of setting P. Suppose that we wish to set P so that PP^* will be about 50 percent. For the four programs that we have studied, the value

$$P = 100/N \qquad (5)$$

has been found to be satisfactory. When N (the total number of addressing references) is larger, then P must be smaller to achieve the same effect. If N is not known, then, of course, P cannot be set by an equation such as (5). If a program is to operate for a given time slice, then N is known. In other cases, it may still be possible to set P by some adaptive process, while monitoring the performance of the system. Further, some fixed value of P may produce good overall performance, since it is not necessary for the algorithm to perform satisfactorily for any particular program, but only on the average.

Assuming the MIN sequence for the four programs, Table II shows the result of using the sequence-dependent algorithm specified by Equation (1) with the parameters set by Equations (4) and (5). For comparison, Table II also shows the result of the sequence-independent algorithm, which may also be interpreted as the sequence-dependent algorithm with $PP_{MAX} = 100$ percent. We see that at $PP_{MAX} = 60$ percent there is little degeneration of performance over the sequence-independent algorithm (i.e., little decrease in the value of PA^* over PA), while at $PP_{MAX} = 50$ percent the degeneration is significant. Using the mean of the results for the four programs, we see that with the sequence-dependent algorithm we can expect to move 45.2 percent of the pages from large-core to core, while we can expect to direct 77.3 percent of all the addressing to core storage. Furthermore, we can guarantee that no more than 60 percent of the pages will be moved.

SUMMARY AND CONCLUSIONS

This paper was concerned with page movement in a directly addressable memory hierarchy. A two-level hierarchy was studied, so the question arose as to which pages should be moved from the second to the first level, and which pages should be addressed in the second level directly. One approach is to move those pages which are used more often, since access time to the first level store is less. The problem is to determine which pages are the most active. Some procedures to detect the high activity pages were developed. These algorithms assume no knowledge of the page activity prior to processing.

The behavior of four programs was studied, and it was shown that some pages were used far more often than were others. For example, it was shown that the 50 percent of the pages which are the most active account for about 95 percent of all the activity. These data on program behavior were then used to test the procedures.

The first procedure was termed "sequence-independent," since although it depends upon the relative number of times each page is addressed, it does not depend upon the sequence in which the addressing takes place. Using this algorithm, it was found that by moving an expected 50 percent of the pages from the

TABLE II—Effect of the sequence-dependent algorithm

PROGRAM		SEQUENCE-INDEPENDENT ALGORITHM		SEQUENCE-DEPENDENT ALGORITHM			
				PP_{MAX} = 60%		PP_{MAX} = 50%	
		PP	PA	PP*	PA*	PP*	PA*
1	INSTRUCTIONS	54.4	81.0	54.0	78.2	48.2	51.9
	DATA	52.3	76.5	52.2	75.4	48.1	51.4
2	INSTRUCTIONS	31.9	76.1	31.9	76.1	31.9	76.1
	DATA	17.4	80.2	17.4	80.2	17.4	80.2
3	INSTRUCTIONS	58.2	82.0	55.9	70.3	48.3	41.1
	DATA	50.0	76.0	50.0	75.9	48.4	68.3
4	INSTRUCTIONS	53.3	85.6	53.0	84.1	47.8	62.6
	DATA	47.3	78.2	47.3	78.2	46.5	73.6
	MEAN	45.6	79.5	45.2	77.3	42.1	63.1

$$P = \frac{100}{N} \qquad K = \frac{PP_{MAX}}{100} \cdot J$$

second to the first level store, one could expect to direct about 80 percent of the addressing references to the first level store. Since the most active 50 percent of the pages account for about 95 percent of all the activity, and since any arbitrary but random selection of 50 percent of the pages would account for an expected 50 percent of the activity, this algorithm may be said to be about two-thirds as effective as it possibly can be.

Sequence-dependent procedures were then studied, with the emphasis on one such algorithm in particular. This algorithm operates just like the sequence-independent algorithm, until a fixed number of pages have been moved. At this point the algorithm shuts down, and no more page movement takes place. Unlike the sequence-independent algorithm, this procedure guarantees that no more than a certain number of pages will be moved. Just as in the case of the sequence-independent algorithm, we found that this procedure can be adjusted to move about 50 percent of the pages from the second to the first level store, while expecting to direct about 80 percent of the addressing references to the first level store. Furthermore, with this algorithm we can guarantee that no more than 60 percent of the pages will be moved.

Thus the page promotion procedures appear to be effective, at least within the limited context in which they have been studied above. Of course, such a procedure would only be one element of a complete hierarchy management algorithm. This complete algorithm must be concerned with many other issues, such as the removal of pages from core storage and the management of page traffic with slower peripherals such as disks. Thus these preliminary experiments are encouraging, but they need to be carried much further, in order that the utility of the page promotion procedures may be judged within the context of a complete system.

REFERENCES

1 D N FREEMAN
A storage-hierarchy system for batch processing
1968 Spring Joint Computer Conf AFIPS Proc Vol 32
Washington DC Thompson 1968 pp 229-236

2 D N FREEMAN J R RAGLAND
The response-efficiency trade-off in a multiple-university system
Datamation pp 112-116 March 1970

3 A L VAREHA R M RUTLEDGE M M GOLD
Strategies for structuring two level memories in a paged environment
Second Symposium on Operating System Principles
Princeton NJ 1969 pp 54-59

APPENDIX I

Suppose that a page is addressed S times, each time using the page promotion procedure described in Figure 2. Define:

λ The expected number of times the page is addressed in the large-core storage.

ϕ The probability that the page will not be moved from large-core storage to core storage.

We wish to show that:

$$\lambda=[(1-P)(1-(1-P)^S)]/P, \qquad (6)$$

$$\phi=(1-P)^S. \qquad (7)$$

To establish Equations (6) and (7), we note that the page will be addressed exactly:

0 times in large-core with probability P,
1 time in large-core with probability $(1-P)P$,
2 times in large-core with probability $(1-P)^2P$,
⋮
$S-1$ times in large-core with probability $(1-P)^{S-1}P$,
S times in large-core with probability $(1-P)^S$.

Then:

$$\phi=(1-P)^S,$$

$$\lambda=0\cdot P+\sum_{s=1}^{S-1}[s\cdot(1-P)^sP]+S(1-P)^S \qquad (8)$$

This establishes Equation (7). The equation for λ must now be reduced further. It is not difficult to show that:

$$\sum_{s=1}^{S-1} s(1-P)^s=[1-P-(1-P)^S]/P^2 -[(S-1)(1-P)^S]/P.$$

Using this result in Equation (8):

$$\lambda=P\left[\frac{1-P-(1-P)^S}{P^2}-\frac{(S-1)(1-P)^S}{P}\right]+S(1-P)^S,$$

$$\lambda=\frac{1-P-(1-P)^S+P(1-P)^S}{P},$$

$$\lambda=\frac{(1-P)(1-(1-P)^S)}{P},$$

which establishes Equation (6).

APPENDIX II

Consider a program with a given $N_1, N_2, \ldots, N_J$. Suppose that a sequence-dependent algorithm is in use and that:

$$P_k=\left[\begin{array}{ll} P & k=0,1,\ldots,K-1 \\ & \\ 0 & k=K. \end{array}\right.$$

There are $N!/(N_1!\, N_2! \cdots N_J!)$ different sequences in which the pages may be addressed. Of these, there is one sequence in which all of the addressing references to page j are performed prior to any of the references to page $j+1$ $(j=1, 2, \ldots, J-1)$. This particular sequence is called the *MIN* sequence. Since $N_j \leq N_{j+1}$ it follows that whenever a page is to be addressed in the *MIN* sequence, a page is selected so that no other page has fewer remaining times to be addressed. We wish to show that of all the possible sequences with a given $N_1, N_2, \ldots, N_J$, the *MIN* sequence minimizes PA.

For a given program, define:

NA The number of times that core memory is addressed.

Since NA is a random variable, it follows that:

$$PA^* = 100E(NA)/N.$$

Thus it will suffice to show that the *MIN* sequence minimizes $E(NA)$.

Define:

α_k The probability that the sequence-dependent algorithm will move exactly k pages from large-core to core.

$E(NA \mid k)$ The expected value of NA, given that exactly k pages are moved from large-core to core.

It then follows that:

$$E(NA) = \sum_{k=0}^{K} E(NA \mid k)\alpha_k.$$

For a given $N_1, N_2, \ldots, N_J$, we will now show that the α_k are independent of the addressing sequence. It will then suffice to show that the *MIN* sequence minimizes $E(NA \mid k)$.

Define:

β_k The probability that the sequence-*independent* algorithm will move exactly k pages from large-core to core.

Then

$$\alpha_k = \beta_k \text{ for } k=0, 1, \ldots, K-1,$$

$$\alpha_k = \sum_{j=K}^{J} \beta_k.$$

For a given $N_1, N_2, \ldots, N_J$, the β_k are independent of the addressing sequence. From the above two equations, it therefore follows that the α_k are independent of the sequence too.

We must now show, for arbitrary k, that the *MIN* sequence minimizes $E(NA \mid k)$. Define:

$\gamma_{j,k}$ The probability that page j is moved from large-core to core, given that exactly k pages are moved.

ζ_j The expected contribution of page j to $E(NA)$, given that it is moved from large-core to core.

Then:

$$E(NA \mid k) = \sum_{j=1}^{J} \zeta_j \gamma_{j,k}. \tag{9}$$

Let us consider ζ_j first. We are given that within N_j addressing references, page j is moved from large-core to core. At each of these references there is a probability P that the page will be moved, if it has not been moved previously. It therefore follows from Equations (6) and (7) of Appendix I that:

$$\zeta_j(1-\phi_j) = N_j - \lambda_j,$$

$$\zeta_j(1-(1-P)^{N_j}) = N_j - [(1-P)(1-(1-P)^{N_j})]/P,$$

$$\zeta_j = N_j/[1-(1-P)^{N_j}] - [[1-P]/P].$$

It can be shown that ζ_j is an increasing function of increasing N_j (if $0<P\leq 1$). Since $N_1 \leq N_2 \leq \cdots \leq N_J$, it therefore follows that:

$$\zeta_1 \leq \zeta_2 \leq \cdots \leq \zeta_J. \tag{10}$$

Note also that ζ_j is not a function of the addressing sequence.

Let us now discuss $\gamma_{j,k}$. The expected number of pages moved from large-core to core is:

$$\sum_{j=1}^{J} [1\gamma_{j,k} + 0(1-\gamma_{j,k})].$$

But it is known that k pages are moved, so:

$$\sum_{j=1}^{J} \gamma_{j,k} = k, \tag{11}$$

where k is a constant.

We are now in a position to complete the proof. Gathering our results from Equations (9) through (11), we have:

$$E(NA \mid k) = \zeta_1\gamma_{1,k} + \zeta_2\gamma_{2,k} + \cdots + \zeta_J\gamma_{J,k},$$

$$\zeta_1 \leq \zeta_2 \leq \cdots \leq \zeta_J,$$

$$\gamma_{1,k} + \gamma_{2,k} + \cdots + \gamma_{J,k} = \text{CONSTANT}.$$

It therefore follows that $E(NA \mid k)$ is minimized by making $\gamma_{1,k}$ as large as possible. Given this, $\gamma_{2,k}$ must be made as large as possible, and so on. It is obvious that this will be accomplished by the *MIN* sequence. Thus the *MIN* sequence minimizes $E(NA \mid k)$, which implies that it minimizes $E(NA)$, which implies that it minimizes PA, which completes the proof.

APPENDIX III

Suppose that the sequence-dependent algorithm is in use, and that pages are addressed in the *MIN* sequence. We wish to devise a method for computing:

$R_{j,k}$ The probability that core storage contains exactly k pages just prior to the time that the addressing of page j begins.

Prior to the first time that page 1 is addressed there are no pages in core memory, so for $j=1$:

$$R_{1,0}=1.$$

Consider now the case when $j>1$. The only way that there can be no pages in core memory just prior to the first time page j is addressed is for there to be no pages in core memory just prior to the first time page $j-1$ is addressed, and for page $j-1$ to remain in large-core. Thus:

$$R_{j,0}=R_{j-1,0}\phi_{j-1,0},$$

where $\phi_{j,k}$ is the probability that page j will remain in large-core when addressed N_j times by the page promotion procedure using P_k. We know from Equation (7) that:

$$\phi_{j,k}=(1-P_k)^{N_j}.$$

For $k\geq 1$ there are exactly two ways in which there can be k pages in core memory just prior to page j being addressed for the first time:

(1) There can be $k-1$ pages in core memory just prior to page $j-1$ being addressed, and page $j-1$ can be moved from large-core to core.

(2) There can be k pages in core memory just prior to page $j-1$ being addressed, and page $j-1$ can remain in large-core.

Thus:

$$R_{j,k}=R_{j-1,k-1}(1-\phi_{j-1,k-1})+R_{j-1,k}\phi_{j-1,k}.$$

When $k=j-1$ this becomes:

$$R_{j,j-1}=R_{j-1,j-2}(1-\phi_{j-1,j-2})+R_{j-1,j-1}\phi_{j-1,j-1},$$

but $R_{j-1,j-1}=0$, so:

$$R_{j,j-1}=R_{j-1,j-2}(1-\phi_{j-1,j-2}).$$

In summary, $R_{j,k}$ may be computed for increasing values of j by using the following equations:

IF $j=1$: $R_{1,0}=1.$

IF $j>1$: $R_{j,0}=R_{j-1,0}\phi_{j-1,0},$

$$R_{j,k}=R_{j-1,k-1}(1-\phi_{j-1,k-1})+R_{j-1,k}\phi_{j-1,k} \quad \text{for } k=1,2,\ldots,j-2,$$

$$R_{j,j-1}=R_{j-1,j-2}(1-\phi_{j-1,j-2}).$$

For $j=1, \ldots, J$ and $k=0, 1, \ldots, j-1$, the $R_{j,k}$ may then be used in Equations (2) and (3) to compute PA^* and PP^*.

Validation of a trace-driven CDC 6400 simulation

by J. D. NOE and G. J. NUTT

University of Washington
Seattle, Washington

INTRODUCTION

A computer center typically faces questions of how to deal with a growing load in the face of tight financial constraints and with the need for lead time for planning ways to meet the demand. The needs may be met by altering the equipment configuration, changing priority algorithms and other features of the operating system, controlling the time scheduling of various classes of job load, or perhaps shifting load from one machine to another if the center is large enough to have such a capability. It is often difficult to gather enough data and insight to show the direction these changes should go and to support the decision to do so. One useful set of tools is provided by simulation backed up by measurements required for validation. However, it is all too common to find that fear of interruptions of the computing center's service to user, combined with an overworked systems programming staff, prevents insertion of the desired measurement probes into the operating system. Then one is restricted to measures that can be derived from the normal accounting log, or "Dayfile", and to software probes that can be injected in the guise of user's programs. In spite of these limitations, useful results can be obtained, and this paper describes validation of a simulation model of a multiprogramming system, the Control Data 6400, making use of these restricted measurements.

The level of detail included in the model affects the degree of confidence one can place on the results of the simulation. It is necessary to represent the system accurately at the level of detail needed to support the predictions of interest. This implies not only enough detail to include the system resources that may be altered, but also enough detail to faithfully represent the interaction of these resources. In a complex system this becomes an important factor since the change in one area may have surprising effects in another part of the system. This argues for inclusion of finer detail.

A counter argument relates to high cost, which is a frequently-discussed difficulty with simulation. However, cost is a relative matter. There is no point in spending money to simulate or measure a system unless the resulting changes lead to savings well in excess of expense; but, when one is dealing with systems costing on the order of fifty to one hundred thousand dollars per month, plus supporting expenses equal or greater to that amount, a relatively small increase in efficiency can more than pay for a useful amount of simulation and measurement effort. It is still important to hold down the cost of simulation and one way of doing so is to avoid more detail in the model than is necessary.

These points, of course, lead to a compromise in the choice of what to include in such a model, and as will be explained later in the paper, we found that the process of validating the model and examining the sensitivity to various parameters gave useful insight on the level of detail that was necessary.

THE CONTROL DATA 6400

Many descriptions of the Control Data 6000 Series exist and there is no point in repeating a lengthy discussion (see Thornton,[1] Control Data,[2,3] MacDougall,[4] Noe[5]). Only the salient features of the system will be described here in order to remind readers of its general structure.

The Control Data 6400 is a multiprogrammed, file-oriented system, using a central processor, ten peripheral processors and twelve channels, a central memory of up to 131K 60-bit words is available to the system and to the user programs. Each peripheral processor has 4K memory (12 bits per word) for its local use. The peripheral processors communicate with the central processor through central memory and they take over many system tasks, such as communicating with the I/O equipment. Concurrent users' programs within the system become files on disk and these files are placed in queues for the needed resources. As the programs are executed, output files are built and transferred to disk for scheduling of output printers and card punches. Programs are executed from central memory by the central processor, using peripheral processors as necessary for subsidiary tasks. A group of eight control points or communication areas is set aside in lower central memory. One of these is unique and is used by the system to keep track of currently unused resources. The others of these may be occupied by users' jobs or by system programs such as JANUS and IMPORT/EXPORT for communication with users' programs. Each control point, 200 octal words in length, stores information unique to the job, such as register contents, program counter, etc., that is needed when the central processor is switched from one job to another.

A variety of operating systems for the CDC 6400 exists. The one on which this simulation is based is SCOPE 3.2 with some modifications that are local to the University of Washington Computer Center. One of these modifications is the disk image (DI) option. This option, when used on a tape request control card, causes the system to first search the disk directory to see if that tape has been copied onto disk. If so, the disk file is used. If not, the operator is requested to mount a tape, which is then copied to disk and retained there for some period of time, (e.g., eight hours). This modification is advantageous in an environment in which many calls may be made for the same tape during any given day, providing disk saturation is not a factor. Another modification is the addition of a staging queue. Jobs that require magnetic tape are held on disk in this queue until the operator has mounted the reel on the tape drive; then they are allowed to proceed into the input queue where they wait for central memory and for a control point.

A feature of the operating system that has been necessary to this simulation and validation study is the system Dayfile. This file accumulates data on individual jobs, such as time into the card reader, time of exit from the input queue, amount of central processor time used, time into the output queue and time off the system. This file, which was originally designed for accounting purposes, provides a significant amount of information that can be used for describing the job load in the trace driven simulator and it will be discussed more fully in subsequent paragraphs.

THE APPROACH

The general approach taken in this study was to develop a simulation at the level of detail that shows the interaction between tasks and system resources. The Dayfile was analyzed to obtain trace data, i.e., to show how the actual job load on any given day made use of the system resources. Validation runs were then made by driving the simulation model with trace data from various days to provide a range of load conditions.

Provisions are included in the simulation model to represent the input job load either as individual jobs (to allow driving the model with the trace data) or as job classes (to drive the model with distributions representative of groups of jobs). Only the former method is reported on in this paper, because it is restricted to a discussion of the model and its validation. The use of job classes will be discussed in a separate paper reporting on use of the model to explore effects of load variations.

This work was done under a number of constraints that limited some of the things we would like to have done. It was important not to interfere with the service being provided by the Computer Center. In particular, it was not possible to insert probes in the system monitor, or to dedicate a peripheral processor to the measurement role, as has been done in work by others (see Stevens,[6] Control Data,[7] and Sherman[8]). As a result, we were unable to include disk statistics among the parameters used to describe jobs, except in a very general way. Disk accesses were not modeled in detail although their overall effect on job flow was accounted for in the simulation through the time interval a job stayed at a control point.

In spite of these constraints, a considerable amount of information was available through the Dayfile and inasmuch as the constraints under which we were operating are not uncommon, it may be of interest to others to see how much can be done without the freedom to alter the system programs for measurement purposes.

DESIGN OF THE SIMULATION

In general structure, the simulation parallels the operating system and allows parameter choices representing a range of available operating system and hardware options. The level of detail of the model is restricted to focus on system resources that may be altered, omitting wherever possible the inclusion of resources for which no decisions may be made. Execution speed of the model is approximately fifty times faster than the real system, i.e., one hour of real system operation is represented by 70 seconds of operation of the model (see Nutt[9] for a detailed description).

The model is written in ANSI FORTRAN and

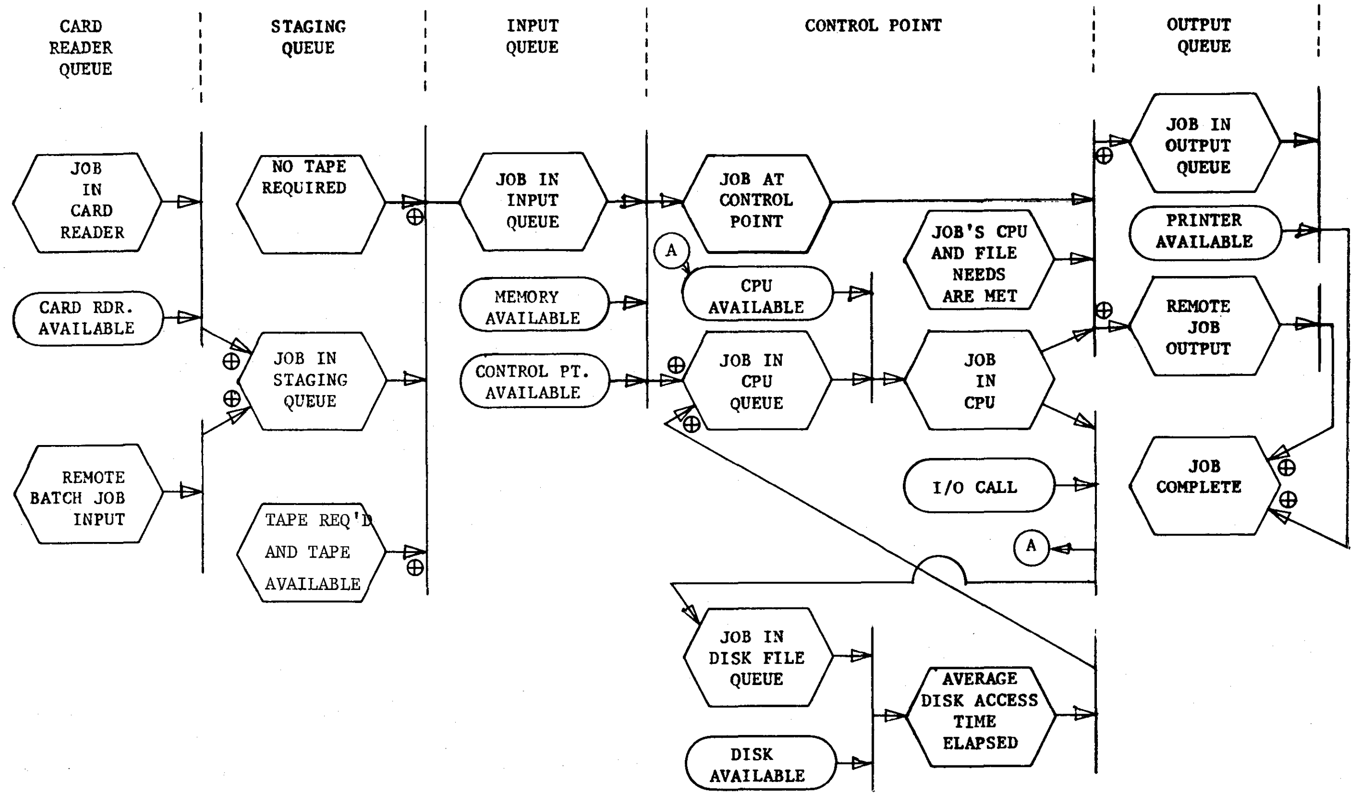


Figure 1—Modified Petri net of simulated CDC 6400

its general structure is similar to BASYS (see MacDougal[10]). ANSI FORTRAN was used rather than a specialized simulation language for two reasons: first it made the program available on both the CDC 6400 and on the XDS Sigma 5 (for which we had no discrete system simulation language compiler); second, since this was done in a learning environment in a university, the use of FORTRAN gave an opportunity to learn more about the details of queue handling and sequence timing that are obscured by the automatic features of simulation languages.

The model is trace-driven, i.e., jobs are represented by the resources they require. It then provides data about jobs that are dependent on system efficiency, such as turnaround, job dwell-time at control points and dwell-time in input and output queues. It also provides data on system performance such as queue lengths, the number of control points in use, CPU utilization, and the amount of central memory occupied by all jobs at sample intervals.

The system actually simulated is shown in Figure 1. The notation is that of the modified Petri net, as introduced in Reference 5. To generate a simulation, it is essential to have a clear representation of the system being modeled. The modified Petri net notation is used because it shows the computer structure under the influence of the operating system.* For the purposes of the present description, it should be adequate to note that the vertical lines are *transitions*. When the necessary input conditions (denoted by incoming arrows) exist, the transition "fires" causing the output conditions to come into being (denoted by outgoing arrows). Arrows marked at either end with the symbol "⊕" denote EXCLUSIVE OR; arrows with no symbol attached denote AND . . . i.e., two or more are required. Note that different symbols may exist at the two ends (input or output) of an arrow.

As an example of the interpretation of Figure 1, "JOB IN CARD READER" represents the existence of one or more jobs in the card reader. Whenever the "CARD READER AVAILABLE" and "JOB IN

* At the state of development of the notation used here, the system representation by Petri nets is qualitative. Further work is in progress to add quantitative features to the modified Petri nets.

TABLE I—Characteristics in Simulator Job Array

Characteristics that are inputs to the model
(Columns in Job Array)

1. Central Memory (during compilation)
2. Central Memory (during execution)
3. Central Processor Time Limit
4. Central Processor Time Actually Used
5. Peripheral Processor Time Used
6. Job Priority
7. Magnetic Tapes Used
8. Disk Image Files Used
9. Number of Cards Read
10. Number of Cards Punched
11. Number of Lines Printed
12. Total Rollout Time
13. Number of Times Job Rolled Out
14. Time Job Entered Card Reader

Characteristics that may be calculated outputs or
(Columns in Job Array)

15. Time of Entry into Staging Queue
16. Time of Entry into Input Queue
17. Time of Assignment to Control Point
18. Time of Entry into Output Queue
19. Time Job Vacated Machine

CARD READER" conditions are true, the associated transition is fired, causing a job to be removed from the card reader and to be placed in the staging queue ("JOB IN STAGING QUEUE"). Note that the staging queue can accept jobs from the card reader *or* the remote input, but not both simultaneously. The use of EXCLUSIVE OR when leaving the staging queue is interpreted as follows: When "JOB IN STAGING QUEUE" is true *and* no tape is required, the next transition fires, altering job status to "JOB IN INPUT QUEUE." *Or* (exclusive) when a "tape job" is in the staging queue *and* a tape is assigned to it, the transition fires.

In making use of the model, it is important to be aware of system resources that are not included. Most important are channels, peripheral processors, and the details of disk access (disk file activity is accounted for only in length of dwell-time of jobs at a control point*). A most important reason for not including channels and peripheral processors is that the number of each available is fixed, i.e., addition of channels and peripheral processors is not among the decisions that might be implemented by the Computer Center. Therefore, it was pointless to examine simulation results dependent on changes in the number of these units. However, it is important to bear in mind that before deciding upon some system changes suggested by the simulator, one should make measurements to see if the proposed change is likely to cause saturation in channel or PPU usage, thereby shifting the bottleneck out of the "range of view" of the simulation.

A principal feature of the simulation model is the method used to describe the job load driving the system. Because this method provided flexibility, it aided validation of the model as well as extrapolation into future situations of interest. Visualize an array in which each row describes a job during the period it is active in the system. As jobs exit from the system, they are removed

TABLE II—Outputs from Simulator

Job Statistics:

Mean and standard deviation for the group of jobs streaming through the simulator are provided for all the characteristics listed in Table 1. In addition, histograms of the distributions are printed for

- Central Memory (average of compilation phase and execution phase)
- Processor Times (central and peripheral)
- Length of time in Rollout status
- Time on Control Point
- Time in Output Queue
- Time in combined Staging and Input queues
- Turnaround Time (from card entry to vacating machine)

Complete trace on each job as it progressed through the model was provided.

Queue Statistics:

- Card Reader Queue
- Staging Queue
- Input Queue
- Central Processor Queue
- Output Queue
- Rollout Queue (i.e., jobs awaiting Rollin)

Resource Utilization: Accumulated busy-time for

- Card Readers and Punch
- Line Printers
- Magnetic Tapes
- Central Processor
- Control Points

* Specifically, an average disk access time was used, as measured by a software probe entered as a user's job. The number of accesses per job was approximated by dividing the job's peripheral processor time by the average access time. The measurement program was written, and the data gathered, by Geoffrey C. Leach, University of Washington.

from the array to provide room for new jobs to be added. Therefore, the array size need be large enough only to handle the currently active jobs rather than all the jobs entered within a given simulation run. The columns in this array represent the trace characteristics of each job, and their contents are listed in Table I. Some of these columns (1 through 14) must be prespecified to describe the job; they may contain positive numbers representing the actual value of the parameter, or may contain negative integers indicating that a value is to be obtained by the simulator system from a distribution identified by the negative integer. This feature allows driving the model with either actual trace data extracted from the Dayfile of the real system, or from data approximated by distributions that represent a job mix, or possibly some combination of the two.

Columns 15 through 19 in the job array show characteristics that may be prespecified (again from actual values or from distributions), or may be calculated by the simulator. This flexibility is helpful during validation of the model since it allows one initially to drive the model with predetermined data, thus force-fitting it to behave like the real system; and then one can back off as confidence is gained, and let the simulator provide more and more of the parameters. This has proven to be a very useful tool during model validation. It is also useful for taking care of "peculiar" jobs, such as control programs that handle terminals and other I/O and which appear to the system as long-term "user" jobs rather than as part of the operating system.

Based upon these input statistics for individual jobs and on the simulator's calculation of the time each job requires to go from point to point in the system, data are gathered on the overall operation of the multiprogrammed system and the nature of the aggregate results provided are listed in Table II.

VALIDATION OF MODEL

General

The procedure followed to validate the model is shown schematically in Figure 2, which starts from the tape containing the Dayfile, i.e., the CDC 6400 normal record of the day's operation, originally conceived as an accounting tool. The Dayfile does not exist in a format suitable to drive the simulator, since entries are ordered according to clock time and information on a unit job is scattered throughout the file. The program DFPREP extracts the proper parameters for each job, creating a new file of properly formatted trace data selected over the time period being examined. As DFPREP reads the Dayfile and creates input for the simulator, it analyzes the data to provide statistics on system performance and on the jobs entered into the system. The trace data are used to drive the simulator, entering only the input descriptions of the job into the simulator and asking it to provide simulated system performance.

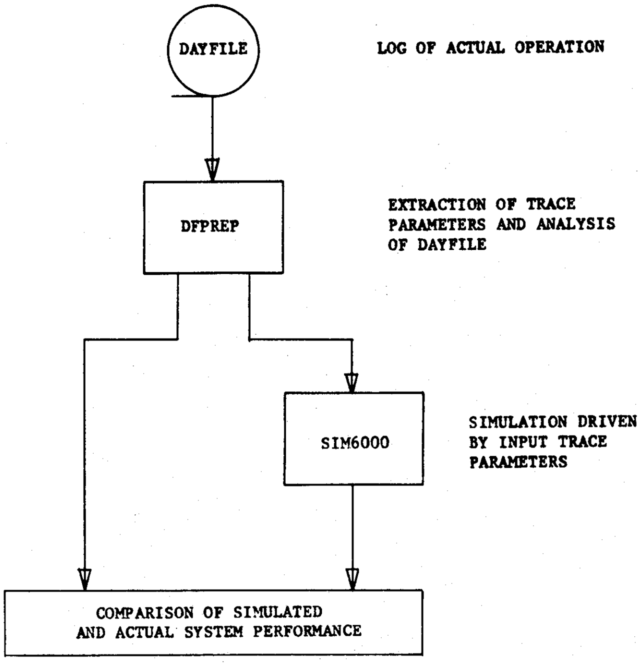

DAYFILE
LOG OF ACTUAL OPERATION
DFPREP
EXTRACTION OF TRACE PARAMETERS AND ANALYSIS OF DAYFILE
SIM6000
SIMULATION DRIVEN BY INPUT TRACE PARAMETERS
COMPARISON OF SIMULATED AND ACTUAL SYSTEM PERFORMANCE

Figure 2—System validation procedure

The process just described sounds deceptively simple. In actual fact, getting the simulated model to agree with measured performance over a range of parameters and job mixes proved to be a complex and time-consuming process. The authors suspect that this is a typical situation and accounts for the fact that the extensive literature describing simulation of computer systems is rather sparsely populated by papers on validation.

On the other hand, this validation phase proved to be very valuable, not only in terms of the essential point of building confidence in the realism of the simulator's performance, but also in the better understanding of the system and of the model that was gained during the validation process. For example, during validation it became clear that it was necessary to extend the model to include jobs that bypassed part of the system, e.g., a remote batch capability that bypasses the card reader queue and the output queue had

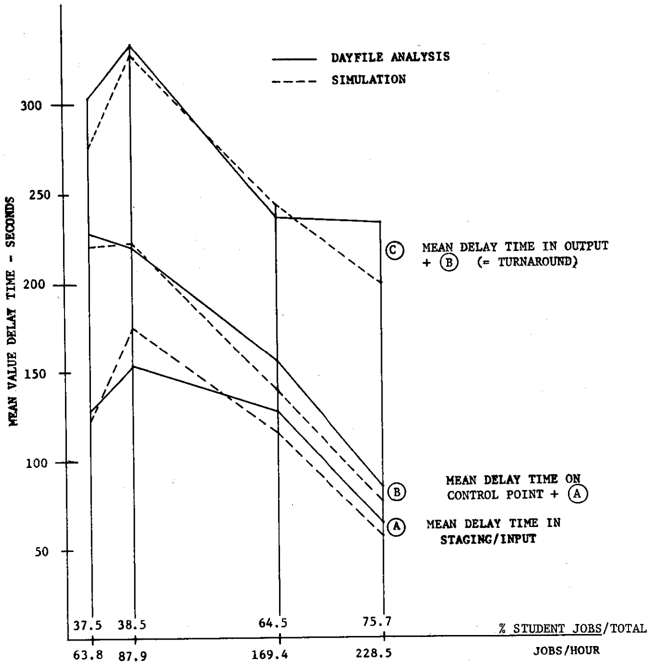


Figure 3—Comparison of simulated and actual job delays

grown in importance in actual system use so that it was necessary to incorporate it. Also, the sensitivity of the system to the manner in which magnetic tapes were handled became apparent as did the effect of the disk image (DI) option. Some errors in the model, in the control point area and in central memory field length variation, were detected and corrected as a result of validation. It was found necessary to approximate the number and duration of disk accesses in order to properly fragment each job's use of the CPU. It also became apparent that it is important to be able to precondition the state of the simulated system when starting it in the middle of the day's operation. In other words, the start-up transient in the simulation, which seems to be on the order of fifteen minutes to one-half hour of simulated time, can obscure the results if one attempts to simulate two or three hours of mid-day performance without preloading the system.

In retrospect, it is tempting to say that some of the points that needed changing should have been included in the model as originally constructed. Yet, we would argue against going overboard in this direction. The question is one of performance sensitivity to a given system feature. If one initially attempts to include so much detail that all significant factors are likely to be in the model as originally constructed, one is certain to end up with a number of unnecessary details that only serve to lengthen the program writing, debugging, validation, and execution time. The authors believe it makes far more sense to construct a model along reasonably simple lines, containing flexibility for injecting needed changes, then adding features when the validation process shows them to be necessary.

Interestingly enough, some of the most difficult things to validate in the model involved human actions and human judgment, particularly the time required to fetch and mount magnetic tapes, and the criteria used by the operators in deciding when to place a program in rollout status (i.e., removing it from contention for core memory and for the CPU) were subject to wide variations. On the other hand, inclusion of these factors was necessary for realism and will provide ways to test some operational procedures and policies that might be applied.

Validation data

Validation and alteration of the model finally converged to a point where reasonable confidence in the model was established. To illustrate this, data are presented that cover four different job mixes and job densities chosen to represent two extremes in computer usage and an intermediate case. One extreme was near the end of an academic quarter, during which the system was used heavily by students trying to finish problems and term projects. The other extreme was between academic quarters when the student load had decreased significantly, leaving a predominance of research jobs and system development jobs running on the machine. The mixes ranged from a low of 37.5 percent student jobs up to a high of 75.7 percent student jobs. The job densities covered a range greater than 3.5

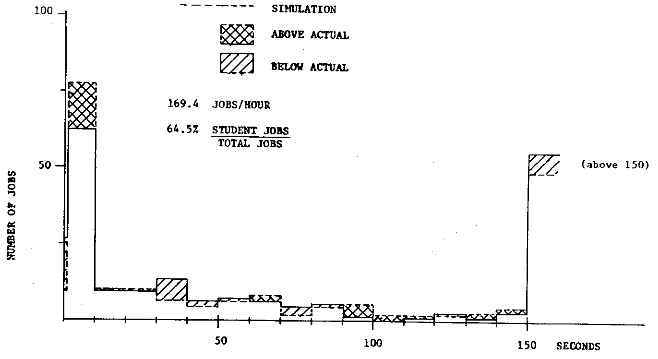


Figure 4—Job dwell time in staging plus input queue

to one in jobs per hour, and included densities near the current record day for this installation.

Figure 3 summarizes the mean values of delay times as the jobs progressed through the system. It shows comparison values for the real and simulated system for each of the four load conditions. The lower curve (A) in Figure 3 shows average delay times through the combined staging and input queues. The next higher curve (B) adds control point dwell times to the staging and input delays. The top curve (C) adds output queue times to the cumulative delays and thus represents turnaround times, i.e., time between entering the card reader and ejection from the line printer.

A word of warning about Figure 3: Do *not* interpret it as "how delay time varies as a function of rate of job flow"; instead, interpret it as "for the specific cases (job mix and density), how do the real and simulated systems compare?" The reason for this is that the higher densities shown in this figure correspond to cases in which a higher proportion of student jobs were run. These jobs on student account numbers are predominantly short jobs with three or four pages of listing and less than ten seconds central processing time (see Hunt[11]).

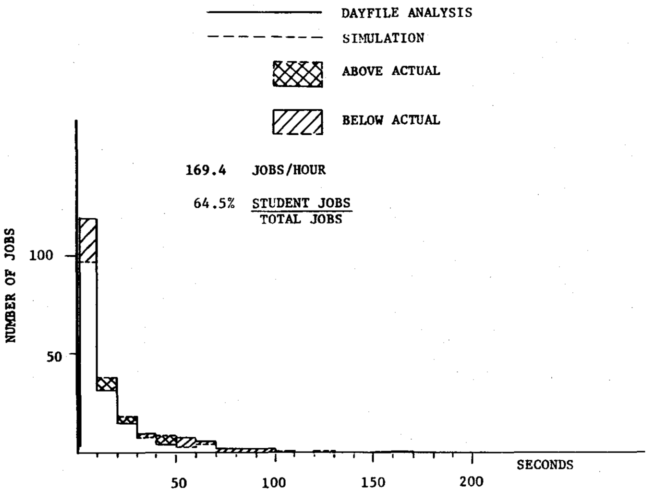


Figure 5—Job dwell time at control point

The set of points corresponding to the highest job density (and highest student job proportion) proved initially to be very troublesome. Upon closer investigation the difficulty proved to stem from tape jobs. For some reason the standard deviation in tape fetch and mount time for that day was unusually high (greater than three times the mean value). For the other days, the standard deviation ranged from 1.63 to 2.25 times the mean value. Therefore, for that troublesome day, the tape jobs were pre-specified in the model, to allow validation of its performance on the remaining, more normal job load.

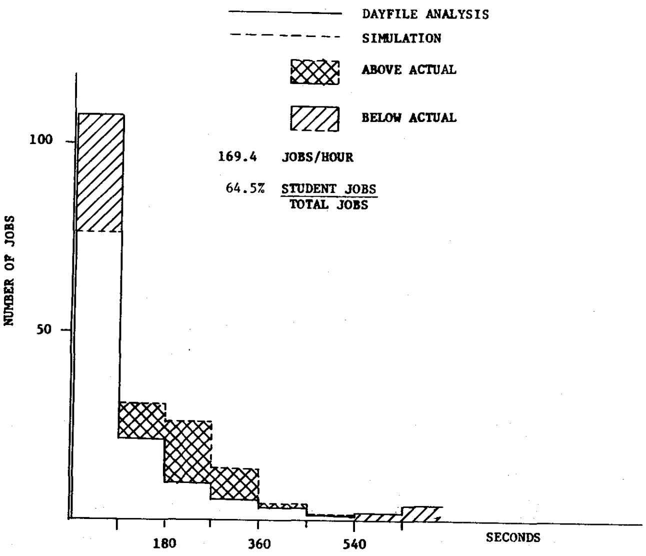


Figure 6—Job dwell time in output queue

In Figure 3 the middle curves (B) are critical, since control point occupancy time directly influences memory and the number of control points available. During validation overall results were found to be most sensitive to agreement in this area. Figure 3 shows only the mean values. The variation in these values is quite great and comparison of the distributions of results is important to the validation. Figures 4 through 7 com-

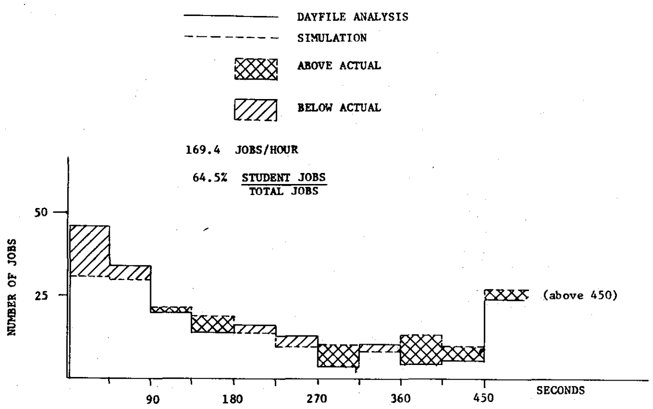


Figure 7—Job turnaround time

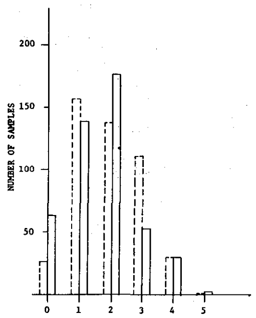


Figure 8—Number of control points in use at sample times

pare the simulated and actual distributions for the various queues and for overall turnaround time for the job load corresponding to 169.4 jobs per hour and 64.5 percent student jobs. The agreement between simulated and actual mean values for this case, as shown in Figure 3, is within 9 percent through the staging and input queue, 11 percent when the control point dwell time is added, and 4 percent for overall turnaround time. The agreement in the distributions, as shown in Figures 4 through 7, shows that the model is operating quite well over a wide range in values. Figures 8 and 9 show similar comparisons between the number of control points and the amount of memory in use in the simulation and in the actual system. These are sample data with the samples not synchronized between the real and simulated systems. A sample was taken approximately every ten seconds and the total run during which comparison was made lasted for two hours.

The agreement of the distribution for this particular load case was representative for the other cases and they are not repeated here. The goal was to verify the model to within 10 to 15 percent. This was generally achieved over the range of loads and over a range of parameters within the model. There are some exceptions, but they are generally understood. For example, the simulation of the 169.4 jobs/hour and 64.5 percent student jobs assumed that there was a constant availability of two line printers. Analysis showed this not to be the case. In the actual system only one line printer was operating initially. Twenty minutes later, another line printer was turned on, and 25 minutes after this event, 3 line printers were operational for a short period of time. Also there were cases where some very unusual system development jobs appeared in the real system and they were not accounted for in the simulation; however, these cases are understood to the point where they do not detract from the confidence in the model to predict performance for the major types of jobs that are important in this environment.

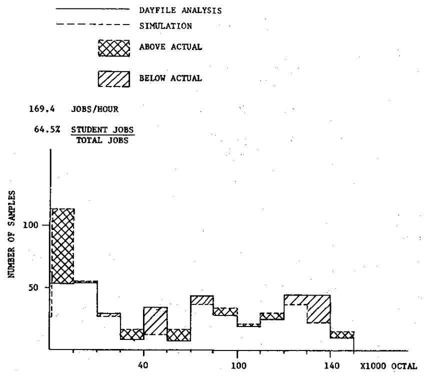


Figure 9—Memory in use at sample times

SUMMARY

At this point, the model is ready for use in experiments to examine the effects of changes in load and system configuration, and these are proceeding.

Perhaps the most important point established so far is that it is possible to gather enough data to validate a useful model, even under the stringent conditions imposed by a non-interruptible computing service center. There is no doubt, though, that special software and hardware measurements would be useful. For one thing, this would provide validation data for a more detailed simulation of the disk system.

The model as it now stands can be used to predict the effects of changes in job arrival density or in job nature. Effects of changes in job CP use and memory use (estimated and actual), number of cards, lines printed, tapes and disk image usage can be observed. Some configuration changes that can be studied include those in central memory, central processor speed or numbers, line printers, card readers, tapes, and number of control points. Scheduling algorithms for these resources can be varied, and the effects of a variety of operational policies can be observed, such as tape-handling, rollout, and control of types and schedules of jobs submitted to the system.

REFERENCES

1 J E THORNTON
Design of a computer: The Control Data 6600
Scott Foresman & Co 1970

2 *CDC 6400/6500/6600 reference manual*
Control Data Publication number 60100000 1965

3 *CDC 6400/6500/6600 SCOPE 3.2 reference manual*
Control Data Publication number 60189400 1968

4 M H MACDOUGALL
Simulation of an ECS-based operating system
SJCC Proceedings 30 pp 735-741 1967

5 J D NOE
A Petri net description of the CDC 6400
Proceedings ACM Workshop on System Performance Evaluation Harvard University pp 362-378 1971

6 D F STEVENS
System evaluation on the Control Data 6600
IFIP Proceedings C34-38 1968

7 *CDC 6400/6500/6600 partner reference manual*
Control Data Publication number 60222500

8 SHERMAN ET AL
Trace driven modeling and analysis of CPU scheduling in a multiprogramming system
Proceedings ACM Workshop on System Performance Evaluation Harvard University pp 173-199 1971

9 G J NUTT
SIM6000 program description and use
University of Washington Computer Science Group Technical Report number 71-07-05

10 M H MACDOUGALL
Computer system simulation: An introduction
Computing Surveys 2 No 3 pp 191-209 1970

11 E HUNT G DIEHR D GARNATZ
Who are the users?—An analysis of computer use in a university computer center
SJCC Proceedings 38 pp 231-238 1971

The evaluation of a time-sharing page demand system

by JUAN RODRIGUEZ-ROSELL*

The Royal Institute of Technology
Stockholm, Sweden

and

JEAN-PIERRE DUPUY

IBM France Scientific Center
La Tronche, France

INTRODUCTION

The techniques of multiprogramming originated in an attempt to better utilize a computer system's resources. Multiprogramming supervisory systems are usually rather complicated and their performance is still poorly understood. Some of the reasons why performance should be analyzed are given in Reference 1. It is possible to monitor the system with hardware devices, but analysis must often wait several hours or days before it can be performed. Also, there are quantities that are impossible to reach with conventional hardware monitoring devices. In particular, process identities are lost. This means that some kind of software monitoring method must be envisaged.

A number of articles[2,3,4,5] have discussed different approaches to software measuring. Since operating systems must compute great quantities of information during their operation, it is clear that the best informed piece of software of the system must be the supervisor itself. In the case of CP-67,[10] the system used at the Institute of Applied Mathematics of the University of Grenoble, a simple approach to monitoring can be used.

CP-67 runs on an IBM System 360/67, which is a machine having the dynamic address translation feature. CP-67 creates the time-sharing environment by generating virtual machines. The real resources of the system (CPU, main memory, channels, etc.) are shared among the users concurrently logged-in onto the system. After a virtual machine is generated the user loads in his virtual computer the operating system of his choice, e.g., CMS, OS/360 or even CP-67. Only the real CP-67 runs in supervisor, non-interruptible mode. All the generated virtual machines run in problem state, and the computer is enabled for I/O and timer interrupts while they are active. Privileged instructions executed by a virtual machine cause a program interrupt to CP-67 which analyzes it and determines the action to be taken.

Of special interest are the paging and dispatching algorithms employed by CP-67.[10] For our purposes it suffices to mention that when processes request the system resources (for example, CPU and main memory) they are divided into two sets, interactive and non-interactive, depending on their type of activity. Those members of each set who have been allocated main memory belong to the active class. When a process causes a page fault, the paging algorithm tries to find a page not in use by someone in the active class. If no page is found satisfying this criterion it will choose the first page it finds, in what amounts to essentially random replacement.

Virtual machines communicate with CP-67 by means of the diagnose instruction.[10] Thus, it is possible for a virtual measuring machine to demand the information contained in special locations within CP-67. The information is updated by CP-67, but the data collection and treatment is done by the measuring machine. The treatment includes writing the data in a disk file and displaying the more important information in a CRT or a console.

The transfer of information by the diagnose instruction and the data treatment are imputed to the measuring machine, which is viewed as just another process

* Presently a visitor at IBM France Scientific Center.

by the supervisor. Furthermore, the virtual machine is dormant waiting for a timer interrupt during the comparatively long intervals between two measurements. The supervisor overhead in running this measuring system is updating the information kept in CP's counters and the process management associated with the virtual measuring machine.

The variables chosen to be observed can be classified as follows:

1. Variables of identification which identify each measure. They are: date, time of day, ordinal number of the present measure.
2. Instantaneous variables. They have a meaning only at the moment at which the measure is taken. There is no relationship between values taken at different times. They are: number of virtual pages in drums and in disks, number of users logged-in, interval between two measures.
3. Cumulative variables. When the operating system is loaded these variables are set to zero. They are incremented by the system. The difference between two values is divided by the time interval between two measures. They are: time in problem state, time in supervisor state, time in wait state, number of pages read per second, number of pages written per second, number of pages "stolen" from the members of the active class per second (see paging and dispatching description), number of virtual machine I/O operations per second, number of CP-67 I/O operations per second, number of privileged instructions per second.
4. Variables of integration. These are variables whose instantaneous value, or even the difference between two values, lacks significance. Instead it is their time average value that is relevant. For example, the number of processes in the active lists may be the same at the moment of two successive activations of the measuring machine. Yet, during this time interval the multiprogramming level has changed often, reflecting the fact that user processes block and become active again. It has been found necessary to form the time integral of the number of users in the active class. This is the only case where relatively many instructions (about 30) have been added to the system's code. The quantities that must be treated this way are: lengths of queues of the active class, lengths of channel I/O request queues.

The results presented correspond to measures taken during the month of July 1971. The system configuration included 512 K 8-bit bytes. Also, two drums used solely for paging, both on the same channel, and two disk units each consisting of 8 disk packs. Each disk unit has its own selector channel. The system measured was CP-67 version 3 level 0. The computer system is run under control of CP-67 from 8:00 to 12:00 and from 15:00 to 18:00. The time interval between measures was set to 100 seconds of virtual time. Because virtual clocks run slower than real clocks this gives real time intervals of between 100 to about 140 seconds. About 2500 sets of measures were collected in the four-week period analyzed in this report.

It is very difficult to give a characterization, even approximately, of the system load. Heavily used components are PL/1, Fortran, Assembler, Lisp and various other languages. Also context editors and debugging aids that run interpretively. Most virtual memories have either 64 pages or 128 pages. Peak loads correspond to about 40 terminals simultaneously logged in, though several of these virtual machines may be running multiaccess systems of their own, such as APL or (virtual) CP-67.

DISCUSSION OF RESULTS

Direct, on-line visualization has proved invaluable in discovering anomalous system behavior. The first lesson we learned is that the system has a tendency to thrash when the number of logged-in consoles exceeds a certain value. The symptoms are a low CPU efficiency and a large number of pages stolen from the members of the active class. Thrashing has been amply discussed in Reference 9 and is well-known to be caused by increasing the multiprogramming level beyond a certain

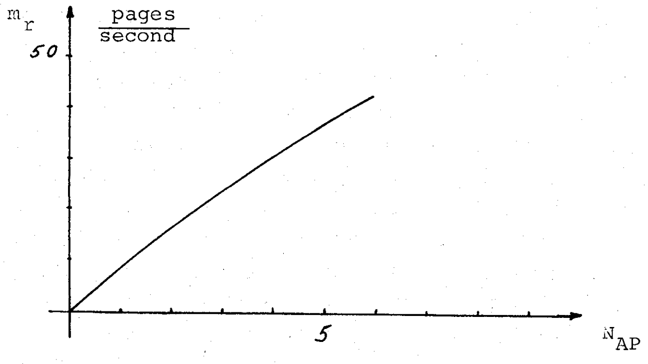


Figure 1—Real page fault rate: m_r
vs.
Multiprogramming level: N_{AP}

point. This causes an increase in the page fault rate, which brings an increase in the traverse time (the time necessary to bring a missing page in memory). This leads to a decrease in total processing time. As far as we know, no experimental data has previously been published on this phenomenon.

There are other conditions whose influence on system behavior would have been very difficult to determine without the CRT display. The supervisor time grows when a virtual machine writes on its console, or, more generally, on a virtual device attached to the virtual multiplexor channel. The supervisor time can easily reach 70 percent or more of the total time when programs write large amounts of data using the multiplexor channel. Another effect which is easily discernible is the degradation in performance brought about by insufficient drum space for paging.

Among the interesting results concerning the system is the low CPU problem state utilization, about 25 percent. Another result is the time the machine spends in supervisor state, which is about 35 percent of the total time. These values are well in agreement with values obtained by a hardware monitoring device, and other values reported in Reference 6. This means that the system, although unbalanced, is not CPU bound.

Some of the results concern thrashing and thrashing conditions. Explanations of thrashing have to do with number of processes actively competing for the resources. Paging rates have been given in References 6 and 7, yet they are given as a function of the number of logged-in users, not as a function of the multiprogramming level.

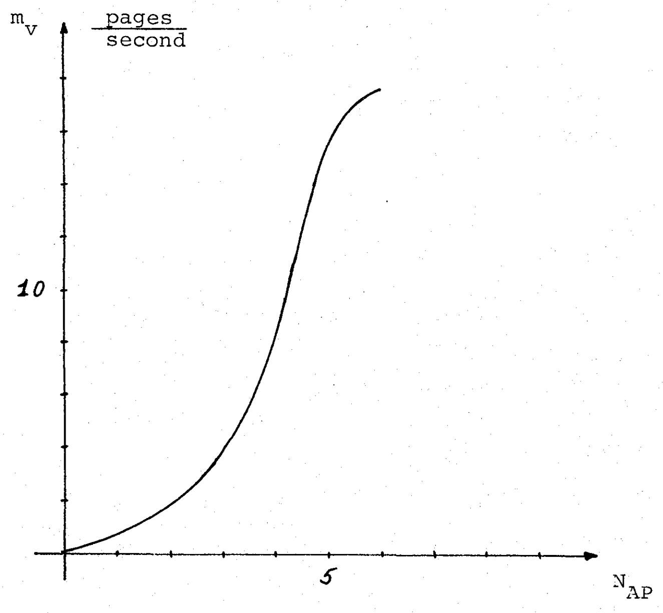


Figure 2—Real stolen page rate: m_v
vs.
Multiprogramming level: N_{AP}

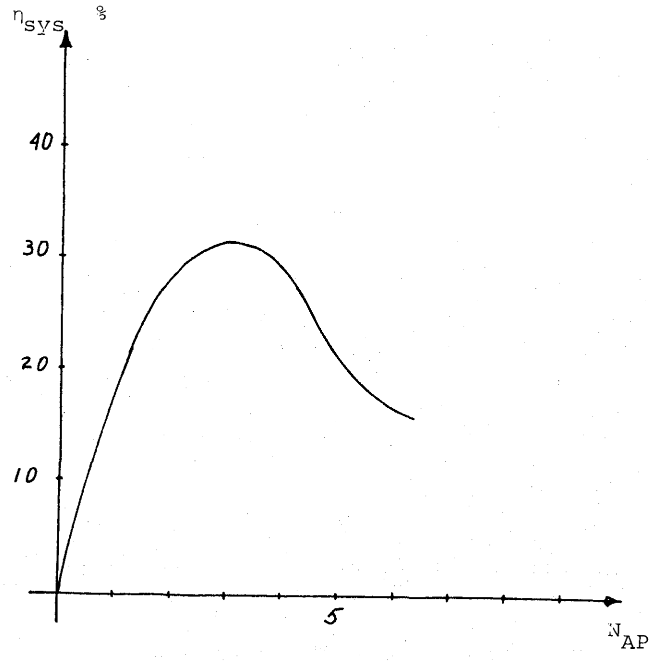


Figure 3—System efficiency: η_{sys}
vs.
Multiprogramming level: N_{AP}

Figure 1 is a plot of m_r, the number of page faults per second of real time, as a function of N_{AP} the mean number of active processes, i.e., the mean multiprogramming level. Figure 2 is a plot of m_v, the number of pages stolen from users in the active class, per second of real time. There is nothing dramatic about m_r, certainly not the steep change that should take place when a certain multiprogramming level is exceeded. Rather, the curve keeps its nearly constant slope in the region shown. Figure 2 indicates that as the multiprogramming level is increased, more and more pages are stolen from the processes in the active class. It has also been found that 70 percent of the pages chosen to be replaced have been modified and must first be written on the drum.

Figure 3 shows the percentage of real time spent in problem state, which we shall call efficiency, as a function of N_{AP}. The maximum efficiency is attained for values of N_{AP} between 2.5 and 4. The efficiency decreases as the multiprogramming level is increased beyond 4. This is well in agreement with thrashing behavior, and it suggests that thrashing sets in at this point. Figure 2 confirms this impression.

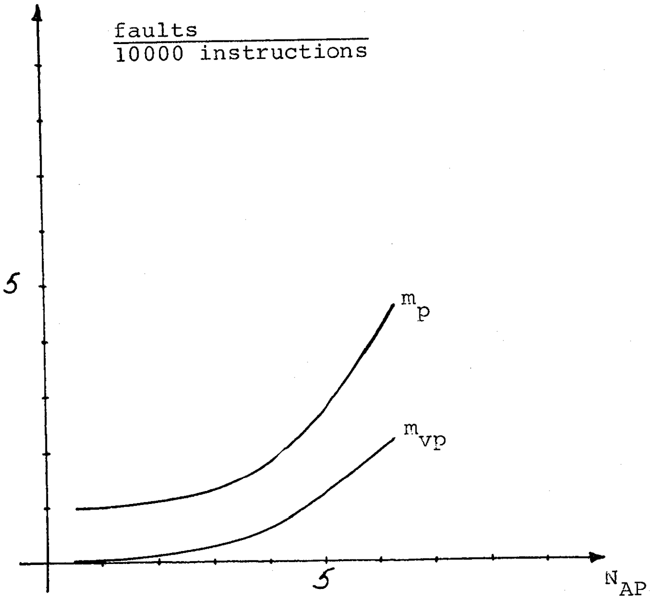


Figure 4—Virtual page fault rate: m_p
Virtual stolen page rate: m_{vp}
vs.
Multiprogramming level: N_{AP}

From Figures 1, 2 and 3, Figure 4 can be deduced. Knowing that the CPU can execute 600 000 instructions per second, Figure 4 gives m_p the expected number of page faults per instruction executed. This curve is the thrashing curve of Denning.[9] The sudden change in slope occurs for values of N_{AP} between 3.5 and 4. Before these values, m_p varies comparatively little. A change in multiprogramming level from 4 to 4.5 brings about a change in m_p four times as great as a change from 3 to 3.5. The lower curve gives m_{vp}, the number of pages stolen per instruction executed. If we subtract m_{vp} from m_p it is apparent that the number of page faults per instruction executed would grow less rapidly if no pages were stolen.

We can conclude that, from the standpoint of paging rate behavior, the system follows what amounts essentially to a working set policy before it gets into memory saturation and thrashing. Afterwards two combined effects take place. One of them is the increase in paging rate because there are, on the average, less pages available in memory for each process. The second effect is due to the paging algorithm that takes the decision of stealing pages which will soon be referenced again. This will make the paging rate still greater.

Any discussion concerning system efficiency must perforce take into account the fact that programs request I/O operations during their execution. If a program executes V virtual time units it will need a total real time of

$$T = V + (Vm_p)T_p + (Vm_{io})T_{io}$$

where the quantities m_p and T_p are respectively, the number of page faults per unit of virtual time and the transfer time of a page fault expressed in virtual time units. A complete discussion of these quantities is found in Reference 9. Similarly, m_{io} and T_{io} are the number of I/O operations per unit virtual time demanded by the process and the time necessary to process one such request. The discussion of the factors affecting these two parameters parallels that of m_p and T_p. It is interesting, however, to notice that m_{io} depends only on the process being considered.

Hence, the process efficiency will be

$$\eta_T = \frac{V}{T} = \frac{1}{1 + m_pT_p + m_{io}T_{io}}$$

In order to simplify the discussion we can assume that T_p and T_{io} are composed only of seek time and/or rotational delay plus data transfer time. We can also include the time spent in the queue waiting for the request to be processed. The overhead in T_p and T_{io} is supposed to be small. Measurement of the paging and disk file channel queues indicate that before thrashing no queues are formed at the I/O channels. Unfortunately this condition does not last long, for when N_{AP} grows the transfer time T_p grows, because longer queues form. There is, in addition, the increase in m_p due to thrashing. In the limit we will have

$$m_pT_p \gg 1 + m_{io}T_{io}$$

Before thrashing the system efficiency grows linearly with N_{AP}

$$\eta_{sys} = \frac{N_{AP}}{1 + m_pT_p + m_{io}T_{io}}$$

As the multiprogramming level is increased longer queues form at the paging channel. If the drum is organized as a FCFS drum, like in CP-67, there will eventually be $N_{AP} - 1$ requests in the queue, each request requiring T_p time units to be served. If T_D is the time the channel is engaged serving one request we would have

$$T_p = (N_{AP} - 1)T_D + T_D = N_{AP}T_D$$

The quantity m_p can be expressed as $A + B\ N_{AP}$, A, B constant.

Consequently,

$$\eta_{sys} \simeq \frac{1}{AT_D + BT_DN_{AP}}$$

And the efficiency decreases hyperbolically with N_{AP}. We can then distinguish three regions in the response of the system's efficiency to an increase of the multiprogramming level. In the first region it behaves nearly linearly, in the third region it decreases hyperbolically and in the in-between region it will grow slowly until reaching the maximum possible efficiency (see Figure 3).

The influence of the multiprogramming level on system efficiency has been discussed in Reference 11. However, that model is exceedingly simplified and neither the variation in paging rate nor I/O operations are taken into account. But it is pointed out that some optimum multiprogramming level must exist. Some experimental data are given in Reference 8 but the CPU is near saturation at the measured points.

Proceeding in the same manner as for m_p we find that m_{io} is practically constant. Its value has been found to be

$$m_{io} = 0.1 \times 10^{-3} \text{ requests per instruction}$$

Typically the time necessary to process an I/O request is about 50 000 instructions. Since no queues are ever found at the disk channels the product $m_{io}T_{io}$ can be assumed to remain essentially constant.

Figure 5 shows the experimental values for n_{WD}, the number of requests waiting to be served by the drum channel, as a function of N_{AP}. To obtain a mathematical model of the system is a more difficult question. Most feedback queuing models assume exponentially distributed service times, which is not the case of the drum. Another model is given in Reference 12 where the drum's service time can be a constant or a random variable. However, in this model file I/O operations are not taken into account. The model discussed in Reference 13 does take into account disk I/O operations, but the I/O request sequence and operations are fixed. Thus it is of no avail in our case. In view of the paucity of existing mathematical models we feel justified in applying the simple M/G/1 queuing model to the drum. The data transfer time of the drum is 3.5 msec for a page and a complete revolution is made in 17 msec. Using the Khinchine-Pollaczek formulae we can find the theoretical curve plotted also in Figure 5. The curves diverge after $N_{AP}=3$, which is about the point where the system begins thrashing.

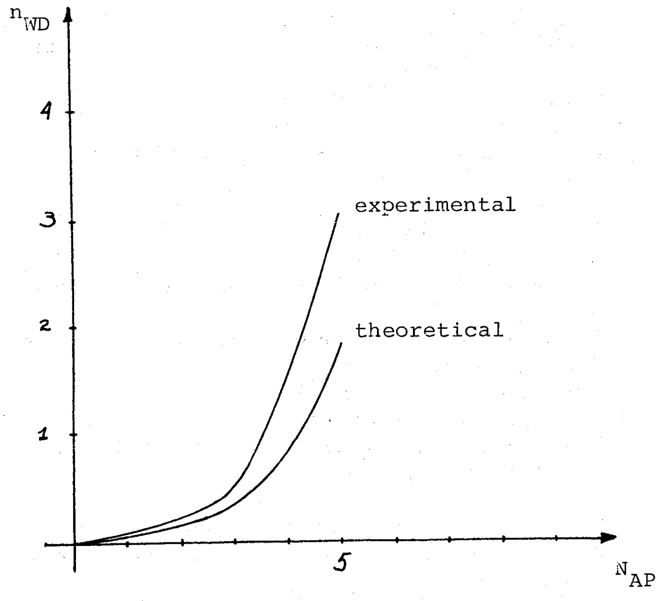


Figure 5—Size of drum wait queue: n_{WD}
vs.
Multiprogramming level: N_{AP}

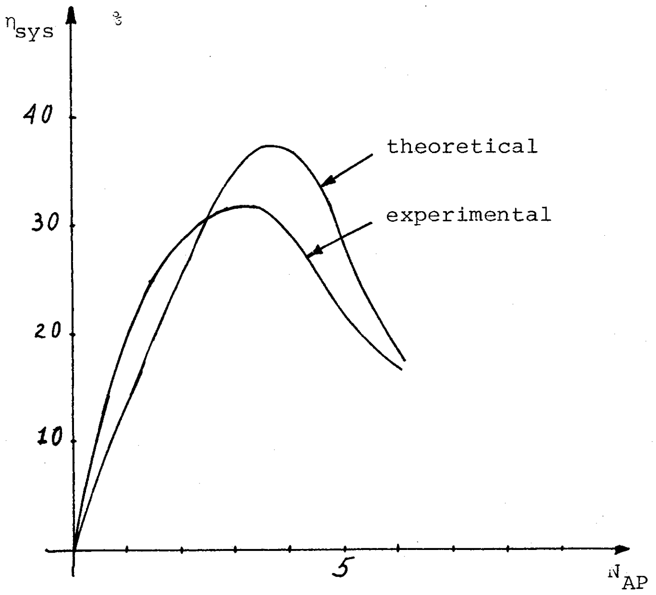


Figure 6—System efficiency: η_{sys}
vs.
Multiprogramming level: N_{AP}

Figure 6 compares the experimental values obtained by using the results of the M/G/1 model for T_p and the measured values for m_p. For small N_{AP} the curve is very sensitive to $m_{io}T_{io}$. Further measurements are necessary in order to obtain better estimates of program behavior. As N_{AP} grows, the number of requests actually waiting in the drum queue is greater than the calculated value and their difference causes the theoretical curve to be an upper bound on the system efficiency.

Figure 7 shows the cumulative distribution function for N_{AP}. The average multiprogramming level never

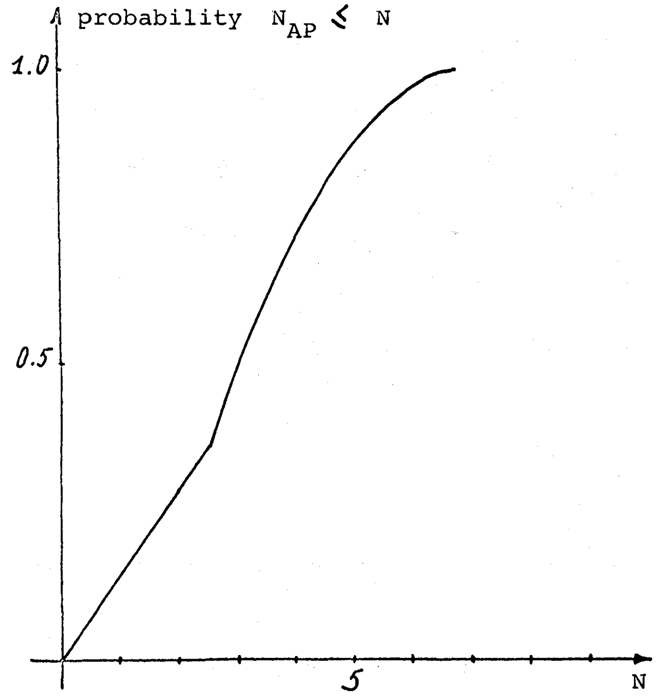


Figure 7—Cumulative distribution function for N_{AP}

reached 7 in this case. All values between 0 and 2.5 have the same probability of occurrence. The most probable value is the same as the mean value and it is 3.0. Assuming the thrashing threshold to be at $N_{AP}=3.5$ we find that the probability of thrashing is 40 percent. There is only a 25 percent probability of being in the optimum region $(2.5 \leq N_{AP} \leq 3.5)$. Of course the multiprogramming level depends on what the users are doing. It seems that 35 percent of the time the multiprogramming level is less than 2.5, meaning that the users are not active enough to impose a heavier load on the system. From the efficiency standpoint it seems that, should the service demand be large enough, it is better to tolerate some thrashing (e.g., $N_{AP}=4.0$) than to be overly anxious to avoid it, imposing, for example, $N_{AP}=2.0$.

CONCLUSION AND FUTURE WORK

The system described permits evaluation of the behavior of CP-67 in a given environment. The limiting factor was found to be main memory, since neither the CPU nor the I/O channels were saturated. Main memory has been increased to 1024K bytes, and the measurement system has permitted evaluation of the new configuration.

Since the original dispatcher does not prevent thrashing, it has been modified to follow a working set policy. Data are being collected to analyze its behavior. In particular, more information is needed about process resource demand and I/O behavior.

ACKNOWLEDGMENTS

The authors are indebted to Mr. M. Peltier and Mr. C. Hans, of IBM France Scientific Center, for their guidance and stimulating criticism. Professor B. Arden, presently visiting Grenoble University, has been kind enough to suggest some modifications. A. Fusi, of IBM Italy Scientific Center, and J. P. LeHeiget, of Grenoble University, collaborated in the initial design of the measurement system. Many friends have reviewed this report and the anonymous referees have provided constructive suggestions.

The work of the first author has been supported by a research grant from IBM Svenska A.B. and a scholarship from the French Government.

REFERENCES

1 D J CAMPBELL W J HEFFNER
Measurement and analysis of large operating systems during system development
FJCC 1968

2 H N CANTRELL A L ELLISON
Multiprogramming system performance measurement and analysis
SJCC 1968

3 A KARUSH
Two approaches for measuring the performance of time sharing systems
Second Symposium on Operating Systems Principles Princeton University 1969

4 B W ARDEN D BOETTNER
Measurement and performance of a multiprogramming system
Second Symposium on Operating Systems Principles Princeton University 1969

5 J H SALTZER J W GINTELL
The instrumentation of Mult cs
Second Symposium on Operating Systems Principles Princeton University 1969

6 Y BARD
CP-67 measurement and analysis: Overhead and throughput
Workshop on System Performance Evaluation Harvard University 1971

7 S J MORGANSTEIN J WINOGRAD R HERMAN
SIM/61. A simulation measurement tool for a time shared demand paging operating system
Workshop on System Performance Evaluation Harvard University 1971

8 S R KIMBLETON C G MOORE
A probabilistic framework for system performance evaluation
Workshop on System Performance Evaluation Harvard University 1971

9 P J DENNING
Resource allocation in a multiprocess computer system
Tech Report MAC-T -50 MIT Project MAC

10 *CP program logic manual*
Form GY20-0590-0 IBM Corp Technical Publications Department

11 V L WALLACE D L MASON
Degree of multiprogramming in page-on-demand systems
CACM June 1969

12 Y C CHEN G S SHEDLER
A cyclic queue network model for demand paging computer systems
IBM Research Report RC 23498 1969

13 G S SHEDLER
A queuing imodel of a multiprogrammed computer system with a two level storage system
IBM Research Report RJ 836 1971

LSI and minicomputer system architecture

by L. SELIGMAN

Data General Corporation
Southboro, Massachusetts

INTRODUCTION

The direct impact of Large Scale Integration (LSI) on minicomputer system architecture has been and will continue to be evolutionary and incremental, not revolutionary.

LSI has been applied to minicomputers most effectively in the form of an incrementally improved technology. When well understood, it has offered predictable cost reductions and performance improvements. It has been most successful commercially when combined with other factors such as improved core memory technology and imaginative new approaches to minicomputer packaging. While the combination of these factors has been steadily driving prices down and performance and reliability up, we are not likely to see the bottom drop out of prices. There will not be a full scale minicomputer in every gas pump in the near future; nor will there be such dramatic improvements in minicomputer performance that minis will replace large System 360s one for one.

INDUSTRY MATURITY

Although technological advances, including increased use of LSI, have improved both price and performance, the recent maturing of the industry has had a more dramatic impact on the minicomputer market; thus, it has also had an indirect impact on the technological base of the industry.

A few years ago, there were many companies selling minis. The cost of entering the market was low, and many of these companies enjoyed some initial success. In the last few years, however, price competition has increased significantly, and the number of manufacturers has shrunk. Concurrently, the OEM minicomputer market has grown, such that, while there are many fewer companies in this business today, those who remain are each manufacturing thousands of units a year. These large minicomputer manufacturers are necessarily becoming very manufacturing-oriented. Their production methodology is growing to resemble that used in producing consumer electronics. Indeed, as Figure 1 shows, a modern mini with 32,000 16-bit words of memory and several peripheral controllers is about the same size as a stero receiver.

Volume production of such a complex product has been facilitated by the improved reliability provided by the new technology. In fact, the greatly increased reliability of the modern mini, based in part on the use of LSI, has been a major factor in its proliferation, especially in real-time systems.

The basic cycle of a maturing industry—larger market, fewer manufacturers, better product, higher volume, increased manufacturing orientation—has naturally driven prices down and opened new application areas.

The indirect impact of LSI grows out of the business cycle as each of the minicomputer manufacturers remaining use many more integrated circuits. The successful manufacture of complex integrated circuitry requires high volume to obtain the high yields required to achieve profitability. Without large volumes, LSI manufacture is burdened with high prices, low yields, stalled development programs, and money-losing sales efforts. However, as minicomputers begin to use more LSI—more because the product design calls for more LSI per unit, and because more units are being made—large IC production develops. At this point the snowball effect begins to work beneficially. There are large circuits requirements, higher yield, lower costs, larger orders. This process is very much in keeping with the evolutionary character of changes in mini price/performance. That is, the manufacturers are increasing their use of LSI technology in predictable, evolutionary way, and not as a means to a breakthrough.

A third element in the growing maturity of the industry has been the diffusion of computer-oriented professionals into most industries. Many of the students

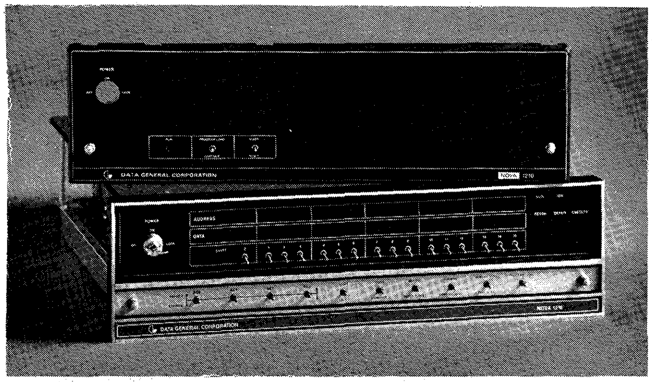


Figure 1—A modern minicomputer with 32,000 words of memory and several peripheral controllers requires only $5\frac{1}{4}''$ of rack space

who had hands-on experience with minicomputers in the university environment will apply this experience to the development of minicomputer based systems for specific applications that had not previously used computers, and they contribute a wealth of ideas to the industry itself.

EFFECT OF LSI

While the result of these technological and business developments has been an evolutionary improvement in the price, performance, and reliability of minicomputers, the effect of the improvement has been revolutionary in one important area—the basic architecture of medium-scale computer-based systems.

There have been two elements in the revolution. First, the minis are real and system designers can trust them. They have become inexpensive, so inexpensive that they are impossible to ignore; and they are reliable, largely because they contain far fewer components and are built in long production runs. For those applications with emphasis on logical decision making, their performance is comparable to large-scale systems.[1]

Thus, system designers are using minicomputers in new applications such as front ends or preprocessors for large business data processing installations; at various levels in hierarchical communications systems; in small, dedicated accounting systems; and in groups in real-time control applications.

The second part of the revolution is more interesting and is developed in detail in this paper. As the minis have become faster, more reliable, and less expensive, they have forced changes in the traditional perception of a computer system. Today most minicomputer processors comprise a relatively small part of the cost of the system. Peripheral devices and their controllers, the system hardware package, and the software to run it all costs as much or more than the basic CPU/memory package.

It no longer makes sense to hold a strictly processor centered view nor does it make economic sense to base all design decisions on a requirement to keep the central processor busy all the time. The low total costs of modern minis make multiprocessor minicomputer systems a practical alternative to large multitask monoprocessors, especially in real-time systems which can be partitioned into individual functional tasks.

MULTIPROCESSOR CONFIGURATIONS

The use of multiple minicomputers, each performing some relatively independent function as part of an overall system, is a natural extension of the concept of the dedicated realtime minicomputer that has evolved over the past several years. This technique is in sharp contrast to the traditional approach to a real-time application, which consists of dropping the entire time-critical package onto a single processor. The alternative approach outlined here substitutes an individual minicomputer for each of the natural functional subsystems of a large-scale application, interconnecting the miniprocessors through an adequate communications path.

The "traditional" or general multiprocessor system is a cross-connected network of processors, memory, and I/O controllers (channels); this configuration

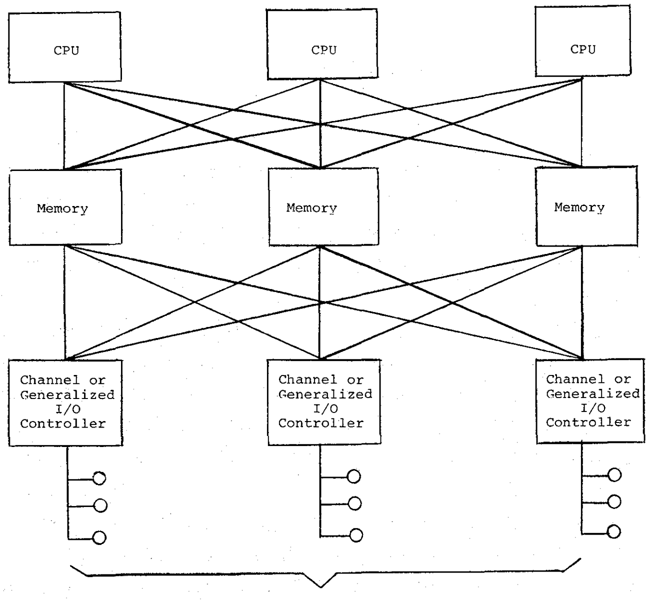

CPU
CPU
CPU
Memory
Memory
Memory
Channel or Generalized I/O Controller
Channel or Generalized I/O Controller
Channel or Generalized I/O Controller

Figure 2a—Traditional multiprocessor

shown in Figure 2, is widely described in the literature[2] and is most often proposed for the multitask environment.

A number of problems have arisen in the completion of the development programs for these systems. Very high utilization is considered imperative for these systems, and a sophisticated multiprogramming executive is required to keep the processor busy. Use of a large executive control program is acceptable so long as the costs of resource allocation and optimization do not exceed the savings which can be attributed to its use. This goal has not always been realized,[3] although a good deal of research has been done on "balanced" resource allocation.[4] Minicomputer multiprocessor systems will inherit similar problems if configured in the traditional patterns. In addition, these problems would have to be solved in light of the particular constraints imposed by minicomputers, especially the amount of software development resources which can be applied economically.

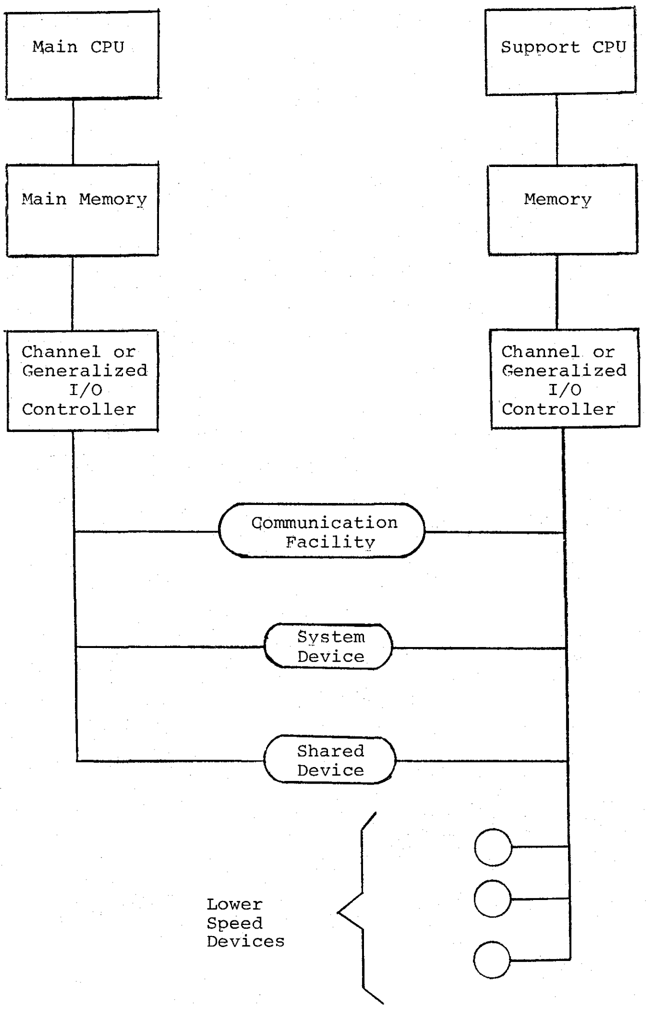


Figure 2b—Associated support processor

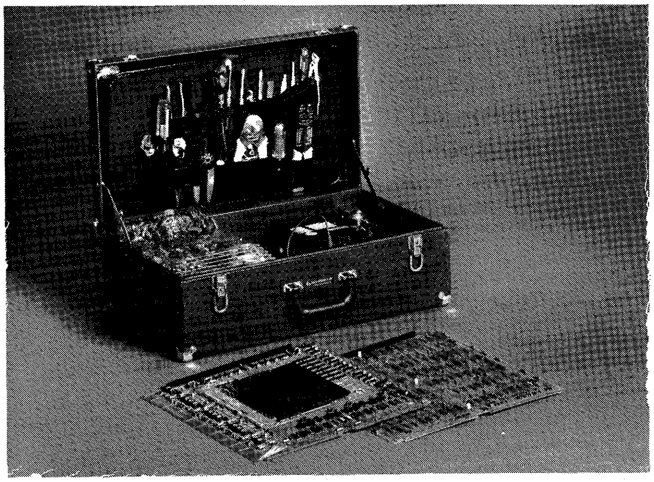


Figure 3—Processor and memory subassemblies

While there are a number of these traditional multiprocessor systems in use today, many more multiprocessor systems are organized in the form of a main processor and one or more smaller support processors. The processor cost in each system is often sufficiently low compared to the cost of the peripherals required to support the data base that it is economical to increase capacity with additional processors.

This associated support processor concept is less elegant, but it has been more successful commercially, mainly because the program development efforts required to make it operational are within the resources of the various manufacturers. The support processors function primarily in the I/O area, freeing the main processor to perform computational functions, while communications, file maintenance, and most I/O activities are handled elsewhere. Even without an explicitly programmed support processor, a typical large scale system is a multiprocessor in that the data channels and some of the peripheral subsystems (especially the disk) contain a substantial instruction processing capacity.

The increased use of LSI imposes some new economic constraints, especially in regards to multiprocessor configurations, on minicomputer systems engineering. The cost of the central processing or memory units are so low that one is forced to view them as modules. In fact, in the case of the Nova 1200 series, as shown in Figure 3, the processor is a single replaceable printed circuit card as is an 8192-word memory module. Other modules, for example a magnetic tape or disk controller, are about equally complex as a processor. Since the manufacturing volume of the processor module is higher than the others, its sales price can be lower. Thus, the use of multiple processors need not add significantly to the system cost.

Once it is decided that a multiple processor approach is the most suitable for a particular application, the intramodule connection facility must be selected. Due to the small physical size and low cost of the modules, the interconnection facility and the mechanical package(s) must be low in cost or much of the potential economic advantage of LSI is lost. Yet, the interconnection facility must support the extremely high combined data rates found at the processor-memory interfaces.

The generalized crossbar interconnection scheme used in the traditional multiprocessor does not meet these needs. Nor does there seem to be a general solution to the multiprocessor system design problem. One does not simply replace a single $200,000 large scale processor with 50 or fewer $4,000 minis, however tempting. However, a number of specialized minicomputer systems have been developed with the concept of the mini as a system element, and some general principles can be abstracted from these designs. As implemented on the minis, none are multiprogramming systems but by necessity they would have been if they had been implemented on a one or two processor large scale machine. They are all organized in a form similar to the associative support processor concept. A processor and sufficient storage to hold its application program and data buffers form a module. Several of these modules are connected via a high speed interconnection bus, so that data and control information can be passed among the several functional programs.

THE PROCESSOR/STORAGE MODULE

The processor/storage module approach of Figure 4 (PSM) calls for the loose interconnection of a number of such modules into a single system according to a partition of the overall application into functional tasks. The modules operate independently but are tied together via an interconnection bus that allows a number of concurrent intermodule logical communications links to be established. Processing is thereby distributed across the system rather than concentrated at a central point.

The logical basis of the PSM approach is "locality", which is the observed phenomenon that the memory address patterns generated by a processor are not random but exhibit a strong clustering. This principle has led to the development of many storage hierarchy systems ranging from cache memories in modern large scale machines[5] to various overlay systems like the chaining feature in Fortran to demand paging timesharing systems. The success or failure of these various approaches depends on the relationship between the amount of locality encountered and the overhead incurred on nonlocal references. The general problem is difficult to model mathematically, although in recent years several approaches show promise.

Locality requires, especially in real time systems, that certain portions of the program be resident in primary (low access time, executable) memory; other portions can be brought in as needed. Minicomputer economics are such that each memory partition might just as well have a processor associated with it. One would then think of the memory as defining a function, specializing a processor for a task just as read-only control memory has been used to define the function performed by microprogrammable controllers.

The interconnected processor/storage module organization described here constrains the programs written to a form that is highly local. Like chaining in Fortran, intermodule references are quite explicit and, hence, visible to the individual programmer who most optimally partition the program.

Not all of the individual minicomputer subprograms need come directly from a partitioning of the task. A PSM can be used as a flexible, programmable controller to enhance the performance of peripheral equipment. In the message switching system example below, the disk control program and associated processor, together with a simplified hardware controller, perform many of the more sophisticated key search functions that would, in a large scale system, be performed in part by the file controller itself. The major advantage of the mini as a controller is that it can be programmed by the user for its particular task rather than micro-programmed at

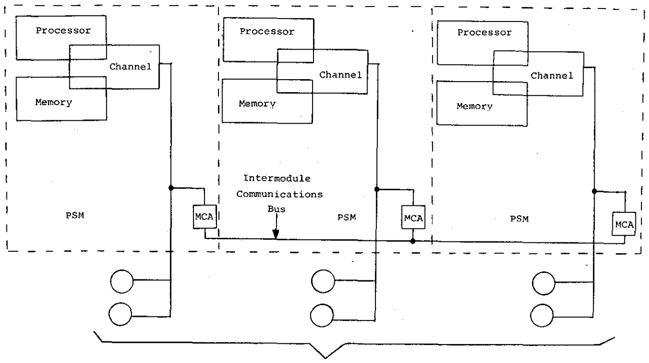


Figure 4—Processor/storage module system organization

the factory for a generalized task. It is far easier to modify a factory supplied operating system than to alter firmware.

In addition, programs can be swapped in and out of a physical memory through the inter-module communication facility. In such a situation, one module would have secondary storage associated with it and would distribute programs on request (or by some scheduling algorithm) to other modules. In essence, it would act as a (micro-) programmable multi-port disk controller serving many processors and would perform certain system level executive functions.

The hardware actually used to interconnect the processor is termed a multiprocessor communications adapter (MCA). One MCA is attached to the I/O bus of each computer in the system, and the adapters are connected together by a common communication bus which is time-division multiplexed among the adapters. Although a single circuit module, an MCA actually contains independent receiver and transmitter subsections, allowing simultaneous reception and transmission of data. Each interface is connected separately to the data channel. The program need only set up an interface for receiving or sending, and all transfers to and from memory are then handled automatically by the channel hardware. A processor with an adapter can establish a link between its transmitter and any receiver it designates, provided that the receiver adapter has been initialized for reception.

A number of logical links concurrently share the single communications bus using time-partioning multiplexer circuitry built into the adapters. If there are N logical links established and communications is proceeding on all, each link receives 1/N of the communications bus time. The bandwidth of the bus is 500 KHz (1 million bytes per second). However, this rate will only be obtained when a large number of links are active concurrently as data rates are primarily determined by the processor's channel facilities. The typical data rate on a single link for a pair of Nova-line computers with high speed data channel feature is 150 KHz.

Each adapter has a unique identifying number assigned to it. These codes are used to specify the number of a second adapter to which the first is to be logically connected. Upon receipt of the first data word from some transmitting adapter, the identifying number of the transmitter is set into the receiver adapter status register; the receiver will subsequently accept further data only from the transmitter, i.e., it "locks" to that transmitter. It must be explicitly unlocked by the receiving processor's program.

A number of the specifications of the MCA arise from the need for the whole system to remain functional despite a hardware or software failure. The transmitter registers must be initialized by the transmitting processor; and the receiver registers must be initialized by the receiving processor. Thus, a transmitter cannot force data into a receiving processor at addresses not specified by the receiver. Transmission of a block terminates when the number of words transferred satisfies the transmitter or receiver with the shorter word count. In addition, a receiving processor is not only protected against a failing transmitter as described above, but the hardware is arranged so that any of the interconnected computers can be stopped or have their power switched off without affecting the other computers still in operation. If a processor adapter attempts to transmit data to an unavailable receiver adapter, a timeout interrupt will occur after a delay of approximately 10 milliseconds. If the receiver is unavailable because it is linked to another transmitter, the transmitter may be restarted for further attempts or the data may be routed to a different receiver.

The size and nature of the data transmission can follow any convention established by the user; no particular structure is forced by the hardware design. In a relatively simple system in which the size and nature of the data block to be transferred is always known in advance, the receiver can simply initialize itself to accept the next block at the completion of the previous transmission.

If the exact size and nature of the data blocks is determined dynamically, a control block specifying the nature of a transfer can be transmitted before the actual data block. With such a convention, the receiver initializes itself to accept a control block of standard format and unlocks itself. The first word transferred to the receiver locks it to the sending transmitter by setting the transmitter's code into its status register, locking it to that adapter transmitter until explicitly unlocked by the program. Thus, once the first word in a control block is accepted, the receiver is locked to that transmitter and can be initialized to accept subsequent data blocks from the appropriate transmitter.

Alternatively, the control block from adapter A to adapter B can be a request for data. Adapter B's transmitter can be started sending the desired data while its receiver is reinitialized to accept a new control block. The hardware itself does not distinguish between a data and a control block.

The first sample system using the PSM approach is shown in Figure 5. The function of the system is small scale message switching and it uses the separate control and data block technique. Separate processors handle: (1) interfacing to synchronous and asynchronous lines, (2) disk queuing, and (3) executive and journal func-

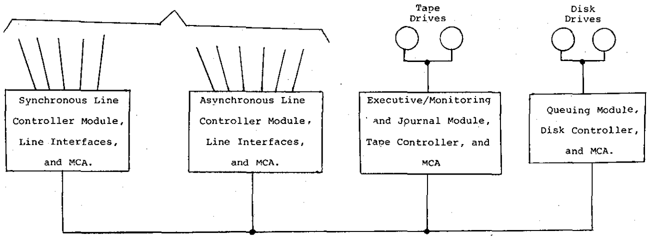

Tape Drives
Disk Drives
Synchronous Line Controller Module, Line Interfaces, and MCA.
Asynchronous Line Controller Module, Line Interfaces, and MCA.
Executive/Monitoring and Journal Module, Tape Controller, and MCA
Queuing Module, Disk Controller, and MCA.

Figure 5—Small-scale message switch

tions. The system is an analog of its large scale predecessors in the sense that PSM's duplicate the functional software components.

The synchronous and asynchronous line control processors duplicate the functions of a transmission control unit as well as the functions of the line-control and polling-sequence subprograms normally run in the central processor. The number of processors required for these functions is determined by the total combined line rate supported. It can vary from one to several depending on the capacity of the rest of the system. Similarly, the disk processor duplicates the key search functions performed by more sophisticated large scale computer disk subsystems. It also decreases effective disk access time since it can use its core (up to 64K bytes) as a data buffer in three ways. First, write operations as seen by the line control processors are virtually immediate, as the data can be passed through the MCA at 300 K bytes/sec. to a primary memory buffer area for future writing on the disk. Second, the queuing processor can optimize the order of seeks to the disk, thereby reducing disk latency. Finally, if sufficient primary memory buffering space is available, the queuing processor can look ahead in the queue for output messages. In some cases, a message accepted by the queuing processor would be output directly from primary memory if it had not yet been written on disk. All of these techniques are possible in a single large scale processor as well as in the multiple mini configuration; however, they are less practical, due to the relatively higher price of the needed primary memory and limited processing time available.

For lines using the bisynchronous communication discipline, the synchronous line processor is definitely needed. The control sequences required by that communication discipline are complex and expensive to implement in hardware. By using software drivers, not only can the control procedures be handled for a large number of lines, but a variety of other line disciplines can also be accommodated (i.e., "foreign" terminals can easily be added to a system.)

The journal and executive/monitoring functions are traditional. As the disk is basically organized as a ring buffer, older messages can be copied from it onto tape during the system's idle periods in order to keep as much space as possible available on disk. A second tape keeps a permanent record, including sequence numbers, terminal ID's, etc., of system activity. The executive function is primarily active in case of error, rebuilding the system after error through journal tapes; it also responds to system level messages, such as requests for retransmission of an earlier message.

The significant point is that new functions and applications modules are not limited by total computational capacity, because additional processor modules can be added as necessary.

A second example of the PSM approach is the experimental multiprocessor small computer system developed at the Plessy Laboratories for the control of a small telephone exchange.[6] The design is inherently extensible and allows for better performance for both voice and data users since the processing capability is distributed among the several processor/memory modules, the number of which can be varied according to the size of the switching network being controlled. The modules are interconnected via a "ring highway" data transmission path similar to the previously described multiprocessor communications adapter.

The organization of the control programs for the multiprocessor complex is almost identical to the organization used in the (essentially) single processor telephone exchange control developed at Bell Laboratories.

In both cases, large tables stored in memory define the state of the switching network itself, and individual, table-driven functional programs are run to implement the various stages in the completion of a call. The major difference between the two approaches is that a spatial distribution of programs across several processors is used in the former, while a time division of the processor is used in the Bell System No. 1 ESS. The amount of interprogram data movement is proportional to the number of calls that can be completed per second. This number is relatively small, since it is limited by the electromechanical nature of the switching equipment. A detailed description of the functions performed is beyond the scope of this paper, but such tasks as dial pules accumulation, network computation, billing, and trunk signaling are typical.

A second important distinction between the Plessy and Bell systems results from different goals influencing the tradeoffs between the costs and benefits of redundancy and error detection. The micro-synchronism of the two computers used in No. 1 ESS allows for the immediate detection of hardware errors at the cost of doubling the amount of hardware required. The PSM approach allows as few as one additional module to be used for lower cost redundancy but does not provide as good error detection. Dual micro-synchronous processors could, of course, be used as modules in the PSM approach which is probably less susceptible to software errors. In addition, many of the real-time problems encountered in a time-division, traditional multitask system can be avoided because modules can be added as the traffic load increases. In fact, as in the message switching system, the modular design facilitates the duplication of a module or application program when use of that program exceeds the available subsystem capacity.

SUMMARY

A number of technical and economic arguments have been presented for the use of multiprocessor minicomputer systems. It is hoped that the examples given will spur the development of a methodology for system design.

REFERENCES

1 A KAY
Computer structures: Past, present and future
AFIPS 1971 Fall Joint Computer Conference Vol 39 pp 395-396

2 F J CORBATO V A VYSSOTSKY
Introduction and overview of the multics system
AFIPS 1965 Fall Joint Computer Conference Vol 27 pp 185-196

3 R J KAY
The management of large scale software development projects
AFIPS 1969 Spring Joint Computer Conference Vol 34 pp 425-433

4 P J DENNING
Resource allocation in multiprocess computer systems
PhD thesis Massachusetts Institute of Technology 1968

5 C J CONTI
Concepts for buffer storage
IEEE Computer Group News March 1969 pp 9-13

6 J R POLLARD
Putting the computer into the telephone exchange
Computer Decisions June 1970 pp 34-37

Approaching the minicomputer on a silicon chip—Progress and expectations for LSI circuits

by H. G. RUDENBERG

Arthur D. Little, Inc.
Cambridge, Massachusetts

INTRODUCTION

Technological progress in semiconductor integrated circuits during the last five years has been truly astounding, and the results of this progress are being aggressively exploited in the designs of new minicomputers. During the 1960s, effort in minicomputer design was concentrated on applying the low price, high speed, and ever-increasing complexity offered by the integrated circuit industries to the original architectures so as to provide greater speed and particularly lower cost. Much of this effort consisted of substituting first MSI and now LSI circuits for the simpler integrated circuits with which previous minicomputers were implemented without greatly changing either architecture or structure and programs. With such substitution, however, tremendous decreases in system cost and some increases in speed were accomplished. Most recently, new structures and architecture are being examined so as to further extend the capabilities as well as economy of minicomputers.

Especially impressive has been the impact of LSI on small, portable electronic calculators; one can readily expect a revolutionary impact soon on minicomputers. For example, during 1969-70, the electronic calculators then sold required five separate MOS/LSI devices. Present calculators function with two or three different devices, and one now being developed performs all calculating functions with a single large MOS/LSI device. In these examples, modern LSI semiconductor technology has created the LSI calculator-on-a-chip, which requires only the input keys, output display, and power supply in order to provide the four major mathematical functions for up to sixteen digits. Soon we shall similarly see the microcomputer and the minicomputer-on-a-chip or possibly two or three chips for separate memory logic and interface. A microcomputer of modest capability using four-bit words has already been conceived and implemented.

Before attempting to evaluate the possibilities and assess the likely impact of recent advances in LSI technology on the minicomputer field, we shall examine the progress made to date and, from this, draw inferences about likely developments over the next several years.

PRESENT ACCOMPLISHMENTS

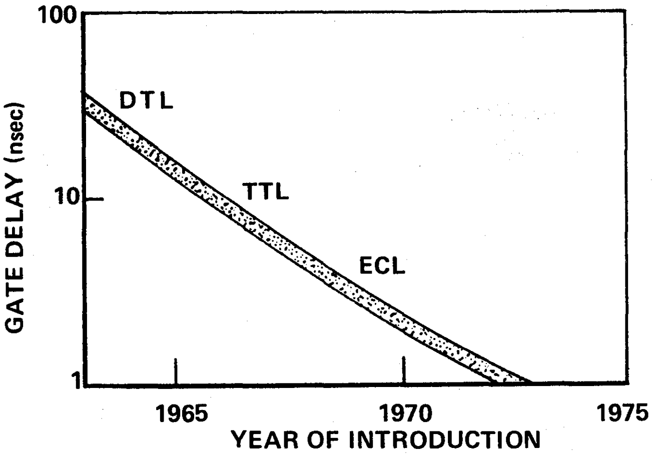
Performance improvements

The great progress achieved by the semiconductor industry during recent years has been widely documented. LSI circuits underwent considerable improvements between 1965 and 1970, as shown in Table I.

TABLE I—Best Characteristics Available in Integrated Circuits or LSI

Function	1965-66	1970-71	Factor of Improvement
Switching delay (nanoseconds)	12	1.2	10:1
Device complexity per chip			
Gate circuits			
Bipolar	12–16	150	1:10
MOS	100	5–10,000	1:100
Bit density			
Bipolar	4	256	1:64
MOS	32	1,024	1:32
Price per device package	$2.00	20¢	10:1
Per gate or circuit function			
Bipolar	$1.00	4¢	25:1
MOS	$0.30	2¢	15:1
Per memory bit (RAM)			
Bipolar	$4.00	6¢	65:1
MOS	$1.00	1¢	100:1

Source: Arthur D. Little, Inc.



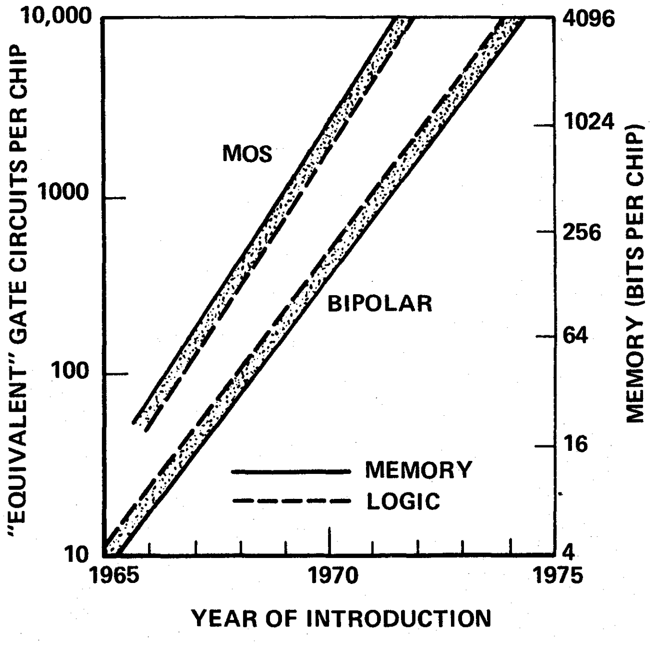
SOURCE: ADL

Figure 1—Speed of bipolar integrated circuits versus year of introduction

These improvements can also be seen from Figures 1–3, which present the achievements in speed, complexity, and price of LSI circuits during this period and project their trends to 1975.

Trade-off relationships

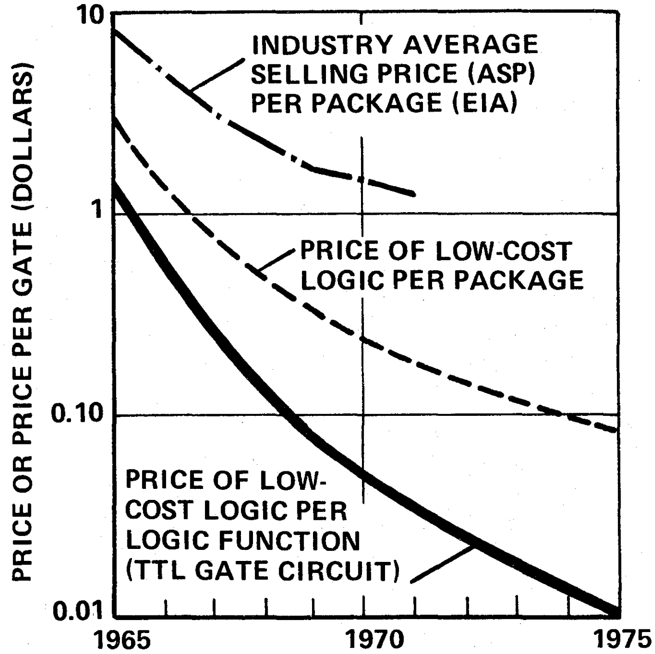
The data of Figures 1-3 and Table I represent peak achievements of devices offered by the industry; naturally, not all of these peak achievements can be obtained simultaneously from the same device. However, by examining commercial offerings more closely, trade-off curves depicting the available relationships between several of these factors can be determined which describe the data available for any technology at a given time.



SOURCE: ADL

Figure 2—Complexity of LSI and IC versus year of introduction

Of considerable interest are the performance trade-offs against price. With increasing circuit function complexity, the price of mass-produced bipolar logic gate circuits will obviously increase due to the decreasing yields obtained for large, complex silicon devices. Ultimately, when device size or complexity measured by the number of gate circuits per module rises into the hundreds, the yields become low and cost of production rises disproportionately rapidly. Figure 4 depicts this relationship for some commonly purchased TTL circuits. These prices represent recent quotations for quantities of ½ to 1 million units per year and are believed representative of the present status.



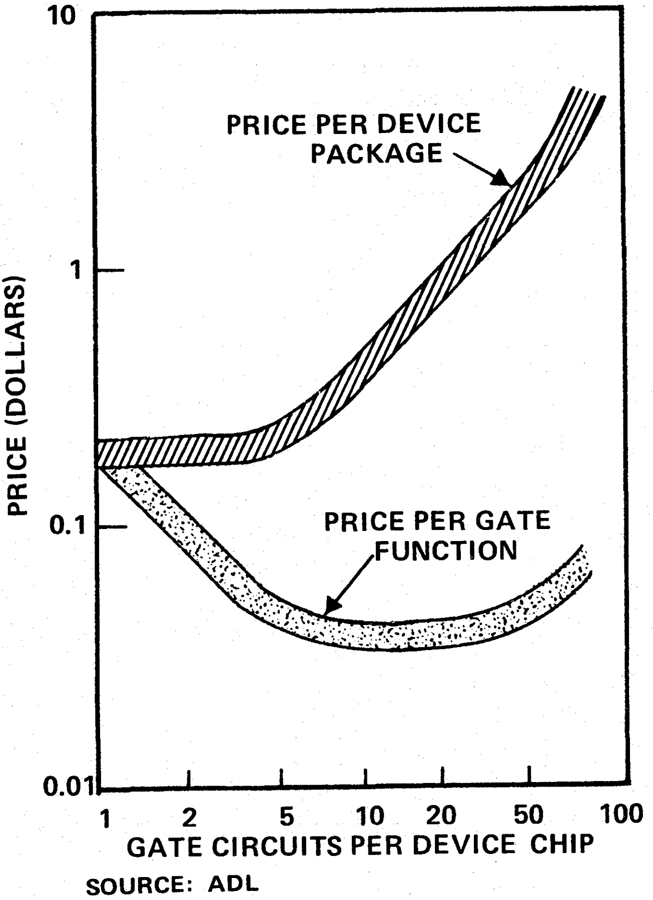
SOURCE: ADL, EIA

Figure 3—Price of bipolar logic circuits

A more useful picture is obtained by dividing the price by the number of gate circuits contained on each chip, thereby obtaining the price per circuit function. In mid-range, this price of bipolar TTL circuits ranges between 3¢ and 4¢ per circuit function. It is readily seen that for devices of low complexity one primarily pays for the packaging; in devices of high complexity, yields become of overwhelming importance and actually raise the price per circuit function.

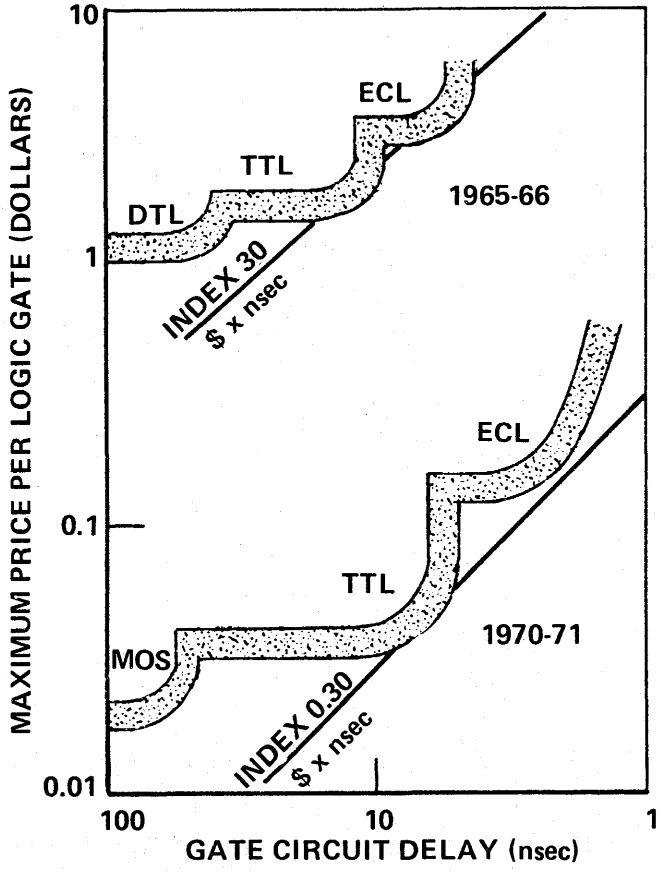
This mid-range price has decreased greatly from year to year and depends, of course, on the volume of purchasing, the type of package (such as military or commercial), and other factors as well as the circuit technology used in implementing the logic functions. Figure 3 illustrates the time development of this economic figure of merit for logic, the price per gate function. In 1965-66, the cheapest logic circuits readily available were DTL circuits, whereas at present the cheapest bipolar integrated circuits are TTL, and some still less expensive logic circuits are MOS. It is seen that the reduction applies to the cheapest units available as well as to the average for the industry; the latter is generally three or four times higher, since it represents also the effect of smaller unit volume of some types as well as military and specialized packaging.

In a like fashion, one may survey the price/speed relationship by plotting such minimum function prices of present offerings against the gate circuit delay or speed (Figure 5). This shows the enormous improvement achieved during the last five years in both lowered prices and faster performance. A similar survey of memory bit prices (Figure 6) also illustrates a dramatic improvement.



SOURCE: ADL

Figure 4—TTL logic price (1971) versus complexity



SOURCE: ADL

Figure 5—Price per logic gate versus speed (delay)

Price/performance index

These classes of devices may be compared by means of a price/performance index. The "present best" figure is obtained by multiplying the price per bit or gate function times the gate or access delay. The result is a price/performance index number representative of the cost per unit speed of the individual logic and memory components. When plotted against time as in Figure 7, it illustrates the dramatic improvement in price/performance achieved by industry during the late 1960s. It would be too much to hope that this

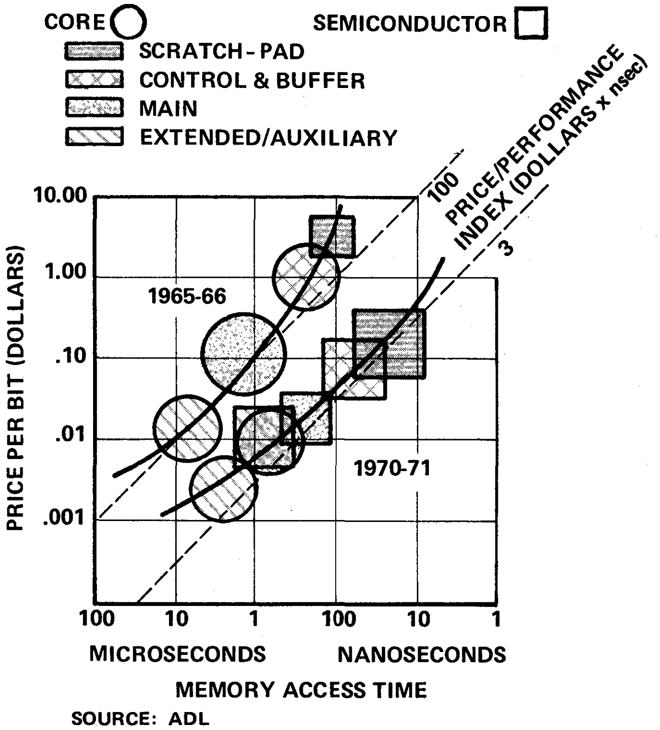


Figure 6—Memory component price versus speed (access time)

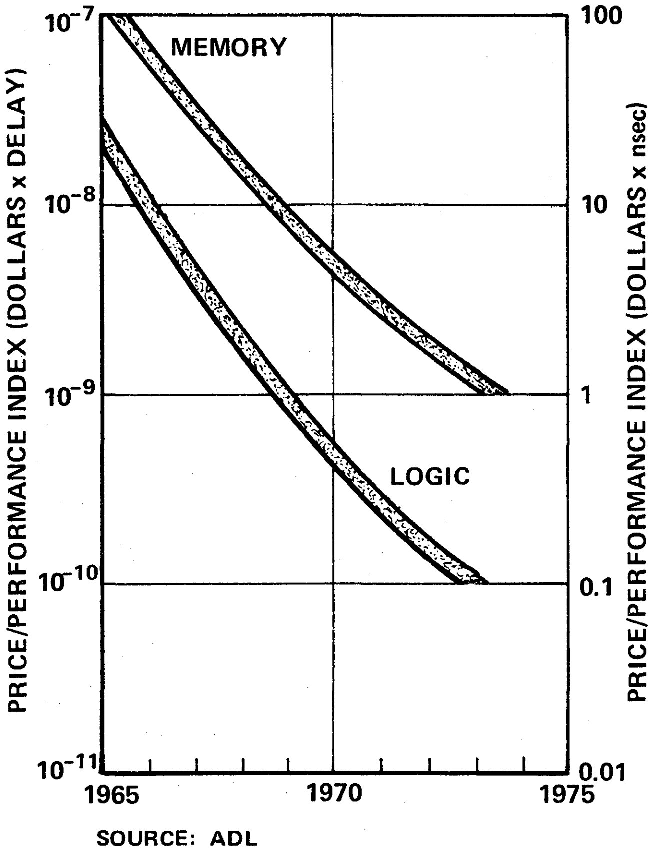
revolutionary trend will continue for the next five years; nevertheless, we expect considerable further improvement on which to base new component offerings, even though we have barely begun to utilize all of the performance now available from LSI circuits.

APPLICATIONS TO MINICOMPUTERS

The earliest applications of LSI have been in logic. This has usually entailed straightforward translation of the transistor circuits of the earliest minicomputers to integrated circuits and now to LSI. Thousands of transistors are normally manufactured by the semiconductor producer on a single wafer and then cut apart and packaged individually, only to be wired together again on a circuit board. In the innovation of integrated circuits, the interconnections are placed over the transistors. The wafers can then be cut into groups of devices that are already interconnected internally. With more recent improvements in semiconductor technology, these groups or circuit functions of the integrated circuit were not cut apart but were further integrated and interconnected on the same silicon chip, leading to large-scale integrated circuits or LSI. The masks also had to be shrunk so that a complex LSI circuit could still be squeezed onto a silicon chip perhaps ⅛ inch square. This development naturally also greatly simplified the assembly of minicomputers and of all other types of electronic systems, as dozens, even hundreds, of packages containing LSI circuits need only to be mounted and further interconnected to comprise a highly capable electronic computing system.

At present, more inventive applications are being developed to capitalize further upon the circuitry which can be included along with the conventional serial logic of an integrated circuit semiconductor chip. One example is the selection, multiplexing, and decoding circuits which permit channeling appropriate outputs of the LSI logic to the several communication buses that interconnect various portions of a minicomputer.

A major part of the circuitry of every minicomputer is concerned with memory. Cores were initially used for this function, but a considerable number of integrated circuits were needed to perform the addressing, drive, and sense functions necessary for utilizing core with a few thousand words. At present, LSI semiconductors are becoming price competitive with core memories



SOURCE: ADL

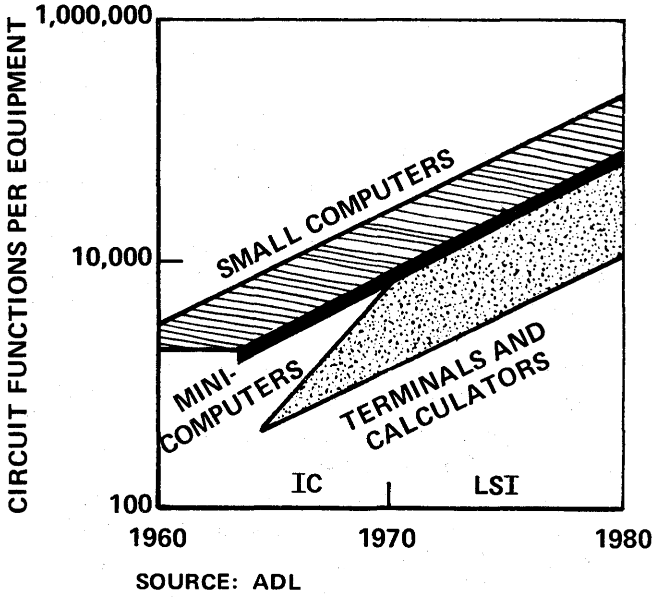
Figure 7—Price/performance index

TABLE II—Implementation of Small Computing Machines with Logic Circuits

	Avg. no. of Circuit Functions			No. of LSI and IC Device Packages	
	Transistors 1965-66	IC's 1970-71	Increase	1970-71	Decrease
Small computer	10,000	30,000	1:3	1,000	10:1
Minicomputer	2,000	5,000	1:2.5	100	20:1
Terminal (simple)	600	1,000	1:1.7	30	20:1
Desk calculator	200	300	1:1.5	3 (soon 1)	66:1 (200:1)

(Figure 6) and with their associated interface circuits for memories of a size suitable for minicomputers. Thus, in the future, semiconductor memories are likely to be used extensively. However, a somewhat different systems organization will be necessary to allow for the refresh and access or cycle timing required for dynamic MOS memories.

Serial shift registers and read-only memories are also being used in addition to random-access semiconductor memories. The most recent trend we have observed is the inclusion of specialized ROM devices to provide fixed microprogramming or look-up tables for special functions and sequences.

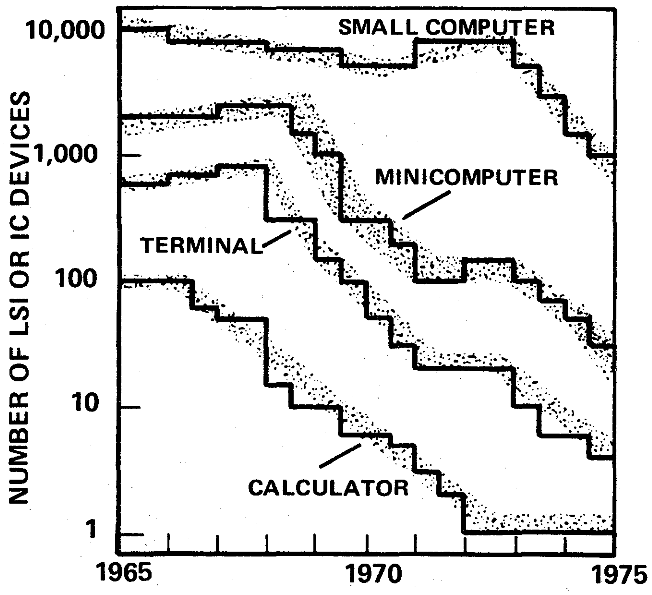


SOURCE: ADL

Figure 8—Functional complexity of computers

The considerable reduction in package count attainable through the use of LSI semiconductors is shown in Table II. Between 1965-66 and 1970-71, while the functional complexity of minicomputers at least doubled, the actual package count of comparable machines decreased by a factor of 20 through the aggressive inclusion of LSI circuits for most electronic functions.

Figure 8 illustrates the number of circuit functions generally utilized in minicomputers and various other classes of equipment at a given time. Of course, considerable variations from this generalization are sometimes encountered. The ensuing count of logic and memory LSI packages is projected into the future in Figure 9. This figure, however, presents a rather optimistic situation on the assumption that available LSI technology is utilized fully with custom circuits. As this assumption is not always economically justified, actual machines may lag this curve and use more but simpler standard integrated circuit devices to achieve interfacing and control with integrated circuits and LSI circuits already available.



SOURCE: ADL

Figure 9—Lower limit of LSI and IC devices in minicomputers

TECHNOLOGICAL IMPLEMENTATION

Since the initial development of simple integrated circuits, a number of processes and circuit technologies have been adapted to semiconductor integrated circuits and more recently to LSI. During most of the decade, bipolar integrated circuits were fabricated using planar diffusion processes. Logic circuits developed from resistor-transistor logic (RTL) to diode-transistor logic (DTL), transistor-transistor logic (TTL), and now, for the fastest speed, emitter-coupled logic (ECL) or common-mode logic (CML). For a considerable time to come, TTL will be the most popular bipolar technology for minicomputers, as it is available in a wide range of devices at reasonable speed and low price. Most recently, several process variations, such as "collector diffusion isolation," are being developed to a practical status to provide lower cost bipolar devices for simpler processes, albeit at somewhat lower speed.

Since 1965 a form of transistor other than the bipolar transistor, termed the metal-oxide-semiconductor transistor (MOS), has been developed for practical applications in silicon devices. As now implemented, the various forms of MOS technology have a smaller cell area for each function and require fewer processes for manufacture (which leads to higher yields), but they are slower than bipolar integrated circuits. Thus, MOS is exceedingly well suited to LSI, and especially to economic use in minicomputers.

Initially, most MOS devices used the P-channel process; now a few memory circuits use the more difficult N-channel process giving higher speed, and a complementary (C-MOS) structure utilizing both N- and P-channels has been perfected. The latter, while somewhat more expensive to fabricate than P-MOS, nevertheless is expected to be cheaper and a little slower than TTL. C-MOS is especially attractive for its low power drain and high noise immunity. Thus, it may well become the preferred device of the future, promising a good compromise in optimizing price/performance trade-offs among different semiconductor technologies.

IMPLICATIONS

The present impact of LSI is a restructuring of the organization and partitioning of minicomputers. To arrive at a useful architecture, the minicomputer must be partitioned according to the availability of LSI circuits and the efficient use of these building blocks throughout the minicomputer, as well as to preserve compatibility with programs previously used. Four directions are observable at this time:

Intermingling of memory and logic. Each LSI semiconductor memory circuit can now be basically self-contained (with address, decoding, drive, and sense circuits where needed) in an economical size. This circuit will also include chip-select for power control or input selection as well as output busing or multiplexing control. Thus, the semiconductor memory circuit can be placed within this system wherever it is needed, not where the economics of core circuits sharing such peripherals dictates.

Utilization of high internal communication speed. Because a minicomputer does not need to operate at the highest speed of large machines, it can benefit somewhat from the speed/price trade-off. The great improvement in price/performance index of both LSI logic and semiconductor memory can provide much more capable small computers containing only a few functional blocks. Furthermore, as this index at a given time is relatively insensitive to the particular speed and performance compromise, a relatively few high-speed memory and logic circuits used serially may be as economical (in terms of component cost) as a larger number of slower components having longer delays. This permits trade-offs between hierarchical memory levels and shared high-speed logic against larger parallel or interleaved logic and memory circuits to obtain a structure most suited for specific application.

Use of read-only memories and programmable ROM's "firmware." This is now the case even in minicomputers. Microprogrammed memories and array logic are especially simple to implement in the short word lengths that suffice for minicomputers. The same reasons of short word length might even lead to consideration of content-addressable memories in future special-purpose computers, whereas such memories are still much too complex for use in general-purpose machines having long words and large instruction sets. In short, the present size of LSI logic and memory arrays is ideally suited to the design of small computing systems.

Further reduction of package count. With the development of a wider range of LSI to implement all functions or subsystems desired for minicomputers, a further drastic reduction of package count can be projected. We anticipate that minicomputers will be designed in the future with as few as 10-20 LSI complex circuits for all memory, logic, and control functions, yet which have capabilities exceeding their present counterparts.

Already we have seen electronic calculators shrunk to a single LSI device. A four-bit microcomputer that uses as few as four different single LSI devices is now available, and we should soon see further applications of semiconductor LSI to the next generation of highly compact minicomputers.

REFERENCES

1 B AGUSTA
A 64-bit planar double-diffused monolothic memory chip
ISSCC Digest of Technical Papers Vol 12 pp 38-39 1969

2 *Annual supplement of computer characteristics quarterly*
Adams Associates Inc Bedford Mass 1968

3 J K AYLING R D MOORE
Monolithic main memory
ISSCC Digest of Tech Papers Vol 14 pp 76-77 Feb 1971

4 J K AYLING R D MOORE G K TU
A high-performance monolithic store
ISSCC Digest of Tech Papers Vol 12 pp 36-37 1969

5 V A DHAKA
Development of a high-performance silicon monolithic circuit
Intern'l Electron Devices Meeting Washington DC
Paper 11.1 Digest p 70 Oct 1970

6 W H ECKTON JR
Development of an ECL gate with a 300 ps propagation delay
Intern'l Electron Devices Meeting Washington DC Paper 13.2 Digest p 100 Oct 1971

7 *Electronic market data book*
Electronic Industries Association Washington DC 1971

8 F S GREENE JR N U PHAN
Megabit LSI memory system
ISSCC Digest of Tech Papers Vol 13 pp 40-41 Feb 1970

9 J A KARP W M REGITZ S CHOU
A 4096-bit dynamic MOS RAM
ISSCC Digest of Tech Papers Vol 15 pp 10-11 Feb 1972

10 W KROLIKOWSKI et al
A 1024-bit N-channel MOS read-write memory chip
Intern'l Electron Devices Meeting Washington DC
Paper 2.1 Digest p 16 Oct 1970

11 T MASUHARA M NAGATA N HASHIMOTO
A high-performance N-channel MOS-LSI
ISSCC Digest of Tech Papers Vol 14 pp 12-13 Feb 1971

12 R N NOYCE
The 4-bit microcomputer
Private communication 1971

13 R N NOYCE
A look at future costs of large integrated arrays
Proc FJCC Vol 29 pp 111-114 1966

14 W M REGITZ J KARP
A 3-transistor-cell 1024-bit 500 NS MOS RAM
ISSCC Digest of Tech Papers Vol 13 pp 42-43 Feb 1970

15 H G RUDENBERG
Large-scale integration: Promises versus accomplishments—The dilemma of our industry
Proc of FJCC Vol 35 pp 359-368 1969

16 H G RUDENBERG
The outlook for integrated circuits
Arthur D Little Inc Chap 3 p 19 1967

The external access network of a modular computer system*

by JESSE T. QUATSE, PIERRE GAULENE and DONALD DODGE

University of California
Berkeley, California

INTRODUCTION

A modular time-sharing computer system, called PRIME, is currently under development at the University of California, Berkeley. Basically, PRIME consists of sets of modules such as processors, primary memory modules, and disk drives, which are dynamically reconfigured into separate subsystems. One ramification of the architectural approach is the need for a medium to accommodate three classes of communications:

(1) those between any processor and any other processor,
(2) those between any processor and any disk drive, external computer system, or other device in the facility pool, and
(3) those between primary memory and any device in the facility pool.

This paper describes the External Access Network (EAN) which was developed for this purpose. The EAN is specialized by certain PRIME implementation constraints. Otherwise, it is adaptable to any system having similar design objectives, or to aggregates of independent computer systems at the same site, which share a similar facility pool and which require system to system communications.

Such systems are computer networks in the sense used by Bell and Newell[1]: that at least two computers, not connected through primary memory, must communicate with each other through messages. Although primary memory is a shared resource in the PRIME system, specific measures are taken to prevent interprocessor communications through shared primary memory pages. It was thought that subsystem integrity would be more easily protected if messages are passed between processors through the EAN. In general, separate computer systems can be interconnected by a network like the EAN if they are reasonably close spacially, close enough to make high data transfer rates economically feasible. For example, the computer network at the Lawrence Radiation Labs at Livermore[2] satisfies this requirement. However, it does not necessarily require the availability and security features of the EAN. One PDP-6 is used as a centralized switching computer, a situation which will be seen to be contrary to the PRIME architectural approach. A system more similar to PRIME in some respects is the Burroughs D825,[3] although it is not a computer network by the definition we have adopted. Primary memory is the medium for interprocessor communications; and an external access network, called the "Automatic Input/Output Exchange", is used for communications between a facility pool and primary memory. The D825 and PRIME share important design objectives, availability being one of them. The similarities are reflected in the external access networks of the two systems. However, there are major differences. They stem from an additional objective of the PRIME system design: to ensure date security to time-sharing users. This leads to protection mechanisms described in the text of this paper and to the fact that the PRIME processors do not communicate through primary memory.

Distances profoundly affect the channel capacity and therefore the functional properties of a network. The EAN is to be distinguished from the many networks designed to operate over great distances. Some similarities do exist in the message formatting, as described for the EAN later in the text. For example, the end of message tag and the source and destination identifier are not new.[4] However, these networks are constrained to lower channel capacities. One very high capacity

* The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government. This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract no. DAHC15 70 c 0274.

long distance network is based upon 1.5 Mbaud links,[5] as opposed to the approximately 20 Mbaud links of the EAN. The ARPA network uses more economic 50 Kbaud links.[6] Another distinctive feature of long distance networks is the way in which availability is achieved. In order to provide routing alternatives, the ARPA network has a much more complex structure than is necessary for the EAN. Also, the ARPA network uses the store-and-forward method to accommodate a non-fully connected structure. The EAN uses a relatively simple structure and has no need to store-and-forward messages.

Thus, the EAN is significantly different from other networks. The difference arises from differing design objectives, principally the high channel capacity, availability, and data security features of PRIME. The design objectives of PRIME, and its implementation, are outlined in the text and related to the EAN. The EAN structure and components are described. Network control sequences are given and the results are summarized.

SYSTEM ASPECTS

The design objectives most important to this paper are high availability and high data security. Availability implies that the system remains in continuous operation, suffering as little performance degradation as possible, during the occurrence of any single hardware or software failure. The PRIME system represents an attempt to achieve high availability without recourse to component redundancy.[7] Instead, the approach is to develop a modular system which utilizes system resources as efficiently as possible, but which is capable of continuous operation whenever individual modules fail. In addition to continuous operation, the system must be capable of maintaining data security in the presence of hardware or software failures. By data security, we mean that the data of any one user is protected from unwanted intrusions by any other user.

The overall implementation is described by another paper in this conference proceedings.[8] In summary, modules such as processors, primary memory, and disk drives, are dynamically reconfigured into separate subsystems while the system is running. One subsystem is designated to run the control monitor which defines subsystem configurations by allocating and scheduling system resources. Its processor is called the *Control Processor.* The others are called *Problem Processors.* Error detection mechanisms, distributed throughout the system, enable the Control Processor to reconfigure subsystems to exclude a faulty module. When the Contiol Processor is suspected of faulty operation, it is automatically replaced by one of the fault-free Problem Processors. The loss of any one module thereby degrades system performance by, at most, the loss of one subsystem.

The fault detection and system reconfiguration capabilities both rely upon two hardware entities: the primary memory map and the EAN. The map serves to partition memory into non-overlapping sets of pages, one set per processor. Subsystems are not permitted to share pages because to do so would considerably complicate the methods used for insuring data security. Instead, interprocessor messages are passed through the EAN where various consistency checks are thought to be more easily implemented. Mechanisms necessary for the isolation of subsystems must be built into the EAN.

An additional objective arises from the fact that the EAN must ensure subsystem integrity while accommodating valid reconfiguration and valid message flow between subsystems. Mechanisms are described later in this paper which aid in the detection and isolation of failures which could lead to the penetration of subsystem boundaries. The subject is treated separately by another paper.[9] Features which ensure availability are described in broad terms here, with many implementation details being omitted. For example, the switching characteristics of the EAN are largely determined by a unit called the Switch Matrix. In order to circumvent catastrophic power failures, each circuit card of the Switch Matrix is powered by a separate supply. The loss of one supply removes one subsystem, but no more. Full details on the Switch Matrix can be found elsewhere.[10]

Other design objectives and PRIME implementation features effected the design of the EAN. Evolvability is generally accepted as a desirable system feature. In our case, evolvability assumes an added significance. PRIME is intended to be an experimental system, a computer laboratory for the study of various implementations of a basic architectural approach. Therefore, the number and types of devices to be interconnected by the EAN could not have been predicted with certainty at the outset. Our strategy was to establish reasonable limits on the growth of the main resources, and to strive for simplicity and generality at the EAN interfaces. Readily available packaging hardware was then chosen which permitted the EAN implementation to meet or exceed the requirements determined by our strategy.

Figure 1 shows the initial configuration of the PRIME system. The main resources are primary

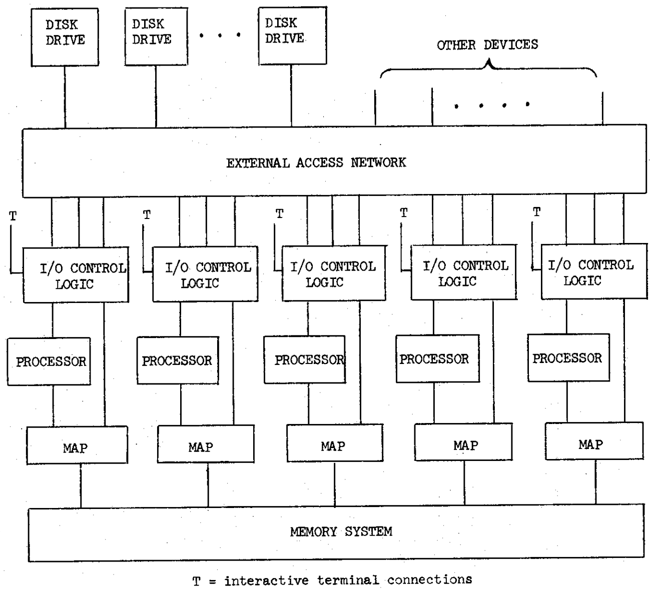


Figure 1—A block diagram of the PRIME system

memory, processors, and secondary storage. In this paper, we group resources in greater detail so that the functional characteristics of the EAN can be better understood. We distinguish between four kinds of resources: primary memory, the processors, the interactive terminals, and the facility pool. The primary memory resource includes the entire bank of MOS memory modules,[11] the memory maps, and the memory part of the I/O control logic shown in Figure 1, although the latter two are physically housed in the processor cabinets. The processor resource includes the five META 4 processors,[12] and all hardware dedicated to the processors (for example, the processor part of the I/O control logic shown in Figure 1). Current plans call for the interactive terminals to be connected directly to the processors, where characters are assembled and relayed to the Control Processor through the EAN. All other resources are referred to in this paper as devices of the facility pool.

The EAN has been scaled to the numbers of processors and devices ever to be found in the PRIME system. Preliminary estimates of the balance of tasks between Problem Processors and the Control Processor indicated that one Control Processor can service approximately four Problem Processors. With an initial system of five processors, a failure in any one results in a total system degradation of 25 percent or less, an availability level which seems acceptable for our purposes. For these reasons, we have permitted our packaging constraints to limit the number of processors to eight. Beyond eight processors, plug-in expansion is no longer possible; but the next eight require only minor redesign. Each processor has been provided with three independent connections to the EAN. Typically, two will be used to control on-going transfers between primary memory and two independent disk drives. The third remains available for interprocessor communications or for communications with other devices within a subsystem. Thus, an eight processor maximum implies a maximum of 24 EAN nodes to be dedicated to processors.

A system with eight processors is estimated to require no more that 20 disk drives of the CDS 215 class.[13] Other devices currently planned for the facility pool include magnetic tape units, a time of day clock, a line printer, and three other computers: an XDS 940, a PDP-5, and a Datapoint 2200. The extensive language repertoire of the XDS 940 on campus will be accessed, at least initially, by means of a 50 Kbaud link to the EAN. The PDP-5 will be used as the processor for a graphic terminal. The Datapoint 2200 is part of the PRIME maintenance and measurement system. Thus, at least 26 devices can be counted. In addition, as will become apparent later, communications between PRIME subsystems require at least one EAN node which otherwise would be used for devices. Therefore, the packaging constraints were permitted to limit to 32 the number of EAN nodes dedicated to devices in the facility pool.

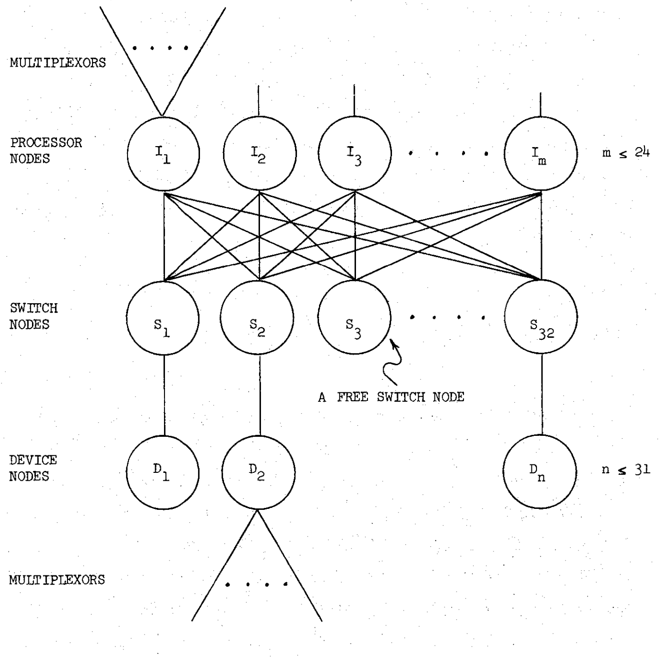


Figure 2—The structure of the EAN

THE EAN STRUCTURE

The EAN has the very simple structure shown in Figure 2. Each path through the EAN consists of two *terminal nodes* linked together by a *switch node.* Terminal nodes serve the purpose of attaching devices to the EAN. A device can be attached to more than one terminal node; but each terminal node is attached to only one device. As explained in a later section of the paper, this device can be a multiplexor which interfaces several devices. Terminal nodes are designated as either *device nodes* or *processor nodes.* Each device node is attached to one switch node. Switch nodes which are not attached to device nodes are called free switch nodes. Each processor node is attached to all switch nodes, but at any one time, at most one switch node can be selected by any one processor node.

A link between a processor node and a device node is established by the processor node selecting the corresponding switch node. A link between two processor nodes is established by both processor nodes selecting the same free switch node. The partitioning into subsystems, and therefore the establishing of links within the EAN, is determined by the Control Processor. However, the dynamic designation of the Control Processor prevents the selection of switch nodes from being done by one centralized control node. Distributing the selection control over all processor nodes makes it necessary for the Control Processor to specify which switch node a Problem Processor should select. Mechanisms are built into each terminal node to guarantee that the Problem Processor has made the correct connection. Therefore the Control Processor retains control of the EAN and acts as if it were the only processor which could establish links. The distinction between processor nodes and device nodes is made for economic reasons. Not all terminal nodes need to provide switch node selection logic and the hardware cost is not negligible.

THE EAN TERMINAL NODES

As is suggested by Figure 3, terminal nodes consist of one, two, or three ports. Each serves a different function. A terminal node is customized to a processor or a particular device by the choice of ports to be included in it. In general, every processor and device is considered to be a collection of registers interfaced to the EAN, usually including at least the equivalent of registers for data input, data output, instruction input, and status output. The interfaces are simplified by providing the *Standard Message Port* (SMP) for register to register transfers. The SMP transfers single words, having fixed length and format. A word may be directed to or from registers within the SMP itself, for purposes of error detection and reporting. All terminal nodes have one SMP.

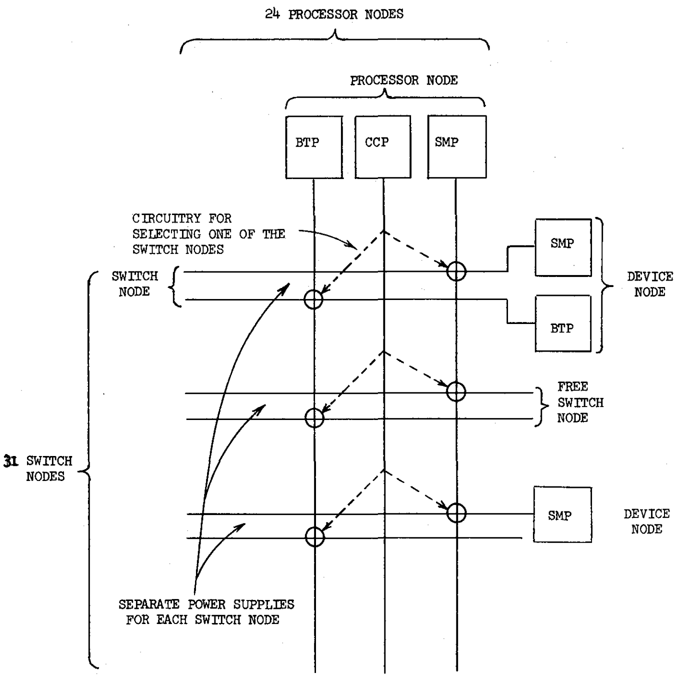


Figure 3—Components of the EAN

A terminal node may or may not include a *Block Transfer Port* (BTP) for transfers between primary memory and devices in the facility pool. The BTP has no fixed word length, transmission format, or error checking capability. Consequently, more complexity is required of an interface to the BTP; but fewer constraints are placed on its design. The purpose is to provide a network channel which has a very high channel capacity and a wide range of operating characteristics. Very high data transfer rates are necessary in order to accommodate disk transfers. A wide range of operating characteristics are necessary because the device controllers of PRIME are not centralized at the devices. In general, the controller logic resides partially at the device and partially in the memory channels. The controller micro-code resides in the micro-code of the processors. Distributed controllers are preferred to centralized controllers for economic reasons. Memory channels and processor micro-code can be shared by all devices.

Each processor node has one *Call Control Port*

(CCP) which is used to select switch nodes, as previously described. The following sections outline the functioning of each port.

Call Control Port (CCP)

A link between a processor and a device is created by the selection of the appropriate switch node through the Call Control Port of the processor node. Two processors create a link to each other by individually selecting the same free switch node through their own Call Control Ports.

The CCP has one more function: to disconnect a processor node from all other nodes. In effect this operation selects a non-existent free switch node.

Block Transfer Port (BTP)

The number of signal lines connecting two terminal nodes must be as small as possible because the number of decoding, driving, and receiving gates increases as the product of the number of processor nodes and the number of device nodes. The BTP was chosen to be 2-bits wide, the minimal width which provides the desired generality and channel capacity. Transmission modes can be full-duplex with one line devoted to each direction, half-duplex with data on one line and clock on the other, or half-duplex two-bit parallel with clock and data interspersed. Maximum transmission rates are expected to be 20 Mbauds total, including clock. This capacity is approximately one-third of the primary memory speed.

In PRIME, a BTP is used to transfer data between primary memory and disk drives of the CDS 215 class. The channel capacity of each drive has been doubled to 5 Mbauds (data) by accessing two surfaces in parallel. One head transfers words with even addresses on one line while the other head transfers words with odd addresses on the other line. These words are accessed from primary memory asynchronously for each line, thereby greatly reducing the effect of the skew between surfaces caused by interchangeable disk drives and disk packs. The half-duplex double frequency transmission mode is used to transfer data to and from the disk drives, in order to maintain head independence all the way to primary memory.

Standard Message Port (SMP)

The SMP accommodates register to register transfers by sending the 52-bit SMP word shown in Figure 4.

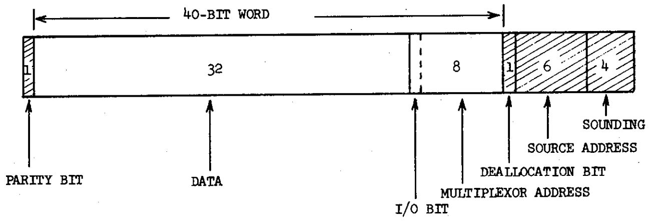


Figure 4—The 40-bit SMP word and 12-bit appendage

As with the BTP, similar considerations led to a 2-bit transfer path for the SMP. Information is transmitted bit serially on one line, using double frequency encoding. The other line is used to receive synchronization signals or messages from the SMP at the other end.

The 40-bit word corresponds to the 40-bit parallel path of an interface to a SMP. The implementation of the META 4 makes 40 bits convenient, although other word lengths would be acceptable. This word includes a data field and a multiplexor address field which suggests the general structure of an interface to the EAN. Terminal nodes are attached to multiplexors which channel SMP messages to addressable registers of the same or of different devices. In effect, the multiplexor address is the address of a register interfaced to a SMP. Registers controlled by fixed sequential circuits can recognize or generate only their own addresses. Registers controlled by processors can deal with any address. The I/O bit of this address is separably distinguishable. It specifies whether the addressed register is an input or an output register. Therefore, the sending SMP can expect a response message within a fixed time duration if the message is addressed to an output register. This feature was added to the SMP for efficiency. Some other timing source would work as well, but more processor time and control storage would be required. The lack of a response is a detectable failure mode.

One of the main concepts involved in the design of the SMP is transparency of the network. This concept is reflected in the fact that anything relevant happening at one port is reported in some way to the other port. This includes synchronization as well as error detection. The shaded fields shown in Figure 4 are generated and used by the EAN for that purpose.

The sounding field precedes the message and is used to alert and synchronize the receiving SMP. An error is detected if the receiving SMP is in transmitting mode. A message received error free causes a signal to be sent back to the transmitter. Another signal is returned to the transmitter when the addressed device picks up the message. The absence of either signal is detected as an error. All errors detected by the SMP of a processor node are reported to the processor as a standard 40-bit message having a special multiplexor address which identifies error messages. All errors detected by the SMP of a device node are transmitted to the SMP of the processor node where they are reported in a similar fashion. Therefore the loss of messages by collision or overlapping is either avoided or reported as a failure mode.

A parity bit is appended by the transmitting SMP and checked by the receiving SMP. In addition, the receiving SMP checks the format of the bit stream (number of clock pulses and time interval between them). These two simple checks, with double frequency encoding, detect a larger class of errors than without double frequency encoding. The probability is extremely small that error bursts will complement an even number of data bits without effecting intervening clock pulses. The failure of any of these checks is reported as an error. It should be noted that we chose to provide checks on a message basis. Longer messages which are made up from several SMP words may require other checks at the process level (like checksums) to guarantee the contents of the total message.

The error mechanisms described above prevent loss of information by distortion of message contents or by message loss. Another mechanism has been introduced to protect against transmission of messages to the wrong terminal node. The integrity of a link through the EAN is ensured by two addresses available at the terminal nodes. Each SMP has its own unique wired-in source address which is appended to each transmitted message. In addition, each SMP has a Source Comparison Register which is loaded as part of the protocol used to establish a link. It contains the source address of the SMP located at the other end of the link. The source address of an incoming message is checked against the contents of this register. A mismatch causes the message to be ignored and an error to be reported.

The Source Comparison Register is cleared by an incoming message having the deallocation bit on. Only the Control Processor can substitute a zero source address for the wired-in source address of the SMP to which it is attached. Thus the clearing of the Source Comparison Register returns the usage of the terminal node exclusively to the Control Processor. The loading of the Source Comparison Register of a device node is possible only with a zero source address message. Therefore the protocol to establish a link between a processor node and a device node requires the intervention of the Control Processor. This mechanism allows the Control Processor to retain control of the EAN and to check that the Problem Processors make correct connections. By the time a Problem Processor is allocated to a subsystem, all of its nodes have selected the non-existent switch node, thereby having disconnected the Problem Processor from the EAN. Thus, a similar loading procedure cannot be used to set the Source Comparison Register of a processor node. Each Problem Processor must load its own register. It does so, through the CCP, as part of the switch node selection operation. In addition, the cleared state of the Source Comparison Register is indicated by a SMP signal which is made available to the device controller. Any controller can have special checking features which distinguish between the Control Processor and Problem Processors.

EAN CONTROL SEQUENCES

A presentation of all EAN control sequences is beyond the scope of this paper. The examples represent typical communications within one subsystem, and between different subsystems. They are simplified by the elimination of steps that are required for higher level system control, but not for EAN control. For simplicity, SMP is not distinguished from the device to which it is attached. Therefore each Problem Processor, PP, has a source address p; a device, D, is identified by its source address d; and the Control Processor, CP, has source address 0.

The first example allocates a device to a subsystem. The Source Comparison Register of a free device is assumed to hold the zero source address reserved to the CP. A device is considered to be allocated whenever its Source Comparison Register holds the source address of a PP.

Allocation of a free device D to a PP:

1—CP connects itself to D.

2—CP loads the address p into the Source Comparison Register of D with a zero source address message.

3—CP receives an acknowledgment from the SMP of D to verify that the Source Comparison Register has been loaded correctly.

4—CP disconnects from D.

5—Eventually, by means of a processor to processor communication, CP instructs PP to connect to D.

6—PP connects to D and loads its own Source Comparison Register with d.

A path between D and PP is now established, and the transmission of messages can begin. The validity of the path is checked by PP with the first message it sends to D. If the terminal nodes of D and PP have a BTP, then transfers to or from primary memory can also be initiated. The form of communication necessary for initializing BTP transfers is generally a one-way stream of messages from PP to D. The memory channel of PP must be independently instructed by PP micro-code.

The next example is a control sequence which establishes a link between CP and PP by request of PP. Two assumptions have been made. First, one free switch node (N) is dedicated to interprocessor communications. (Consequently, only one pair of processors can be in communication at any time.) Second, there is an interrupt channel (I), separate from the EAN, which can be used by any processor to interrupt any other processor.

Initiation of communications by PP:

1—PP interrupts CP through I. This is a request for communication. If possible, PP then continues to run its task while waiting for CP to satisfy its request.

2—When CP is willing to communicate with PP, it connects itself to the switch node N and loads its own Source Comparison Register with p.

3—CP interrupts PP through I.

4—PP then connects immediately to N and loads its own Source Comparison Register with zero.

5—PP sends the first word of the message to CP, thereby notifying CP that the link has been made.

SUMMARY

The PRIME system can be viewed as a spacially small computer network for which the EAN serves two purposes: to interconnect subsystems, and to interconnect devices within subsystems. The architectural approach represented by PRIME is reflected in availability, data security, evolvability, and economic features of the EAN. As a result, the EAN has both similarities and dissimilarities with other networks.

Communications with subsystems are made possible by very high channel capacities, a feature which distinguishes the EAN from spacially large networks. Data security is accommodated by providing for subsystem to subsystem communications through the EAN, instead of through primary memory, and by detecting failure modes which might lead to penetration of subsystem boundaries. Availability considerations, as well as data security requirements, lead to various error detecting and reporting mechanisms which are built into the SMP. Other availability features are the ability of the EAN to exclude faulty modules by reconfiguring subsystems, the ability to redesignate any Problem Processor as the Control Processor, the limitation of the effect of failures, and the network transparency which considerably eases fault diagnosis. Economy considerations lead to the construction of terminal nodes from three types of ports, the provisions for distributed controllers, the simplicity of the SMP interfacing, and the simplicity of the EAN structure. Finally, evolvability is served by plug-in expansion to 24 processor nodes and 31 device nodes, and by the multiplexor address field which permits fan-out from each terminal node.

ACKNOWLEDGMENTS

The authors would like to thank R. S. Fabry, C. V. Ravi, and particularly B. R. Borgerson for numerous discussions and ideas which significantly influenced the design of the EAN.

REFERENCES

1 G C BELL A NEWELL
Computer structures, reading and examples
McGraw-Hill Book Company Inc New York 1971 Chapter 40 pp 504-512

2 *Livermore time sharing system*
LRL Internal Document

3 J ANDERSON S HOFFMAN J SHIFMAN R WILLIAMS
D825—A multiple computer system for command and control
AFIPS Conference Proceedings Volume 22 pp 86-95 FJCC 1962

4 R SCANTLEBURY P WILKINSON K BARTLETT
The design of a message switching centre for a digital communication network
Information Processing 68 North Holland Publishing Co Amsterdam 1969 pp 723-727

5 D DAVIES
The principles of a data communication network for computers and remote peripherals
Information Processing 68 North Holland Publishing Co Amsterdam 1969 pp 709-715

6 L ROBERTS B WESSLER
Computer network development to achieve resource sharing
AFIPS Conference Proceedings Vol 36 SJCC 1970 pp 543-549

7 B BORGERSON
A fail-softly system for time sharing use
P-12/CSR Computer Systems Research Project
University of California Berkeley

8 H BASKIN B BORGERSON R ROBERTS
PRIME—A modular architecture for terminal-oriented systems
AFIPS Conference Proceedings SJCC 1972 Volume 40

9 J QUATSE P GAULENE
Fault protection in the I/O channels of a modular computer
International Computing Symposium Fondazione Cini Venice Italy April 1972

10 D DODGE
Design of the switching matrix for PRIME
Master's Project University of California Berkeley
December 1971

11 B BORGERSON R FREITAS
The PRIME primary memory system
W-17/CSR Computer Systems Research Project University of California Berkeley

12 *Digital scientific META IV series 16 computer system*
Reference Manual Publication Number 7032MD Digital Scientific Corporation San Diego Jan 1971

13 *Century Data Systems—Model 215 disk drive*
Reference Manual RM215 Century Data Systems Anaheim California

14 F HEART et al
The interface message processor for the ARPA computer network
AFIPS Conference Proceedings Vol 36 SJCC 1970
pp 551-567

An over-the-shoulder look at discrete simulation languages

by IRA M. KAY

Southern Simulation Service, Inc.

The past of simulation languages resembles any other history of a subject. For those interested in computers, it presents an absorbing story with moments of outstanding achievement. When Shakespeare said "The Past is the Prologue" he preceded the age of the computer, but he recognized that, regardless of the field, the history, the habits and the education of those engaged in any enterprise shape the way the future will be. Accepting this hypothesis, it then behooves us to look to the past if we are to see what the future holds in digital discrete simulation.

Any practitioner of the art of simulation for a number of years can truthfully say that the past ten years have been both fruitful and full. The term "fruitful" is used in the sense that a relatively new art has been applied to such a wide range of subjects. Mr. Stanley Reed of IBM, in describing the applications to the attendees of the First Annual Simulation Symposium in Tampa, Florida in January 1968 gave a list of applications, and they are repeated here as Appendix I. Any art that can serve this many fields must certainly be classified as fruitful! "Full" is used in the sense that simulation languages have proliferated as tools to serve the researcher. The author, in an invited paper for the Fifth Annual Simulation Symposium provided an inventory of general purpose digital discrete simulation languages. After excluding any special purpose languages, and of course, any digital continuous languages, there were thirty-five inventoried and described. An alphabetical list of the languages (augmented by later additions) is contained in Appendix II, and demonstrates the "fullness" of our past ten years.

One might ask why so many languages have appeared on the scene, and wonder about their availability today. The "why" is hard to explain, other than to say that in many cases pride of authorship kept many going when common sense should have indicated that other available languages would fulfill the needs. In other cases, a simulation compiler was required for a specific computer, and the authors proceeded with what they felt met the needs of the user and thus produced a new language.

Regardless of the numbers, however, there is truly no difference to the philosophy of how the languages work for modeling and simulation. The basic alikeness of all simulation problems is what provides a fertile field for simulation languages. Each in its own manner has a timing routine which keeps scheduled future occurrences in chronological order, each advances a system held variable representing the current time consistent with the occurrences as their scheduled time is reached, and each provides for the gathering of statistics concerning the specific variables (values) the researcher might wish to evaluate. Examination of the languages produced reveals that there were three basic approaches to providing the simulation languages and almost all the languages fit into one of these three styles, or "families".

First, the authors took an existing scientific language like FORTRAN or ALGOL and prepared sub-routines to provide the user with the special requirements of simulation languages, i.e., timing, set manipulation, random deviate generation, statistical summary and input-output. Then the user is able to write in the "host" scientific language, calling the furnished subroutines for the special actions desired. This is a direct method of getting the new user quickly into using simulation languages, in fact, we can almost call it the painless approach, and this ease of transition is the strongest argument for using languages of this type. This writer chooses to name this the GASP family, after the best known of that type language. A "family portrait" of these languages is:

GASP FAMILY

ESP (The **E**lliott **S**imulation **P**ackage)
FORSIM IV
GASP (**G**eneral **A**ctivity **S**imulation **P**rogram)
HOCUS (**H**and **o**r **C**omputer **U**niversal **S**imulation)

JASP (JOSS Conversational Simulation Program)
PA3
PROSIM (**Pro**gram System for **Sim**ulation)
QUICKSCRIPT
SIMON
SIMQUEUE
SPURT (A **S**imulation **P**ackage for **U**niversity **Re**search)
UNS
UNISIM

The second family of languages uses a logical block design for describing the model to be simulated. This means that blocks of certain shapes and identifiers call for designated actions on the part of the computer. Almost in a sign language then, a researcher can portray his model, calling for the generation of transactions, their path through a complex world, the capacities and capabilities of the world through which it must pass, the alternate solutions under blocked overloaded or changed conditions, and still without, or with most minor computer programming capability, the researcher can simulate the world of his model. The compiler interprets the logical design and produces an operational program, adding without demand a wide range of statistical summarizations concerning the period of simulation. This type of family must be called the GPSS family, after the best known and most copied language of all. A "group photograph" follows:

GPSS FAMILY

BOSS (**B**urroughs **O**perational **S**ystem **S**imulator)
EGPS (**E**xtended **G**eneral **P**urpose **S**imulator)
FLOW SIMULATOR
GESIM (**G**eneral **P**urpose **D**iscrete **Sim**ulator)
GPDS (**G**eneral **P**urpose **D**iscrete **S**imulator)
GPS (**G**eneral **P**urpose **S**imulator)
GPS K (**G**eneral **P**urpose **S**imulation **K**)
GPSS (**G**eneral **P**urpose **S**imulation **S**ystem)
GPSS/NORDEN
QUICKSIM

The last family of languages are those which provide a syntax of their own and have the necessary compiler capability to react to the simulation demands of the user for all the aforementioned prerequisites of a simulation language. These type of languages require a programming expertise on the part of the user, but in turn permit him the widest range of operation in dealing with complex problems, and the most ability to tailor his program to fit a particular problem. They are also the most parsimonious in memory use and least costly in running time. This group is called the SIMSCRIPT family after the best known and most widely used of that type. They include:

SIMSCRIPT FAMILY

CLP (**C**ornell **L**ist **P**rocessor)
CSL (**C**ontrol and **S**imulation **L**anguage)
GEMS (**G**eneral **E**lectric **M**arketing **S**ystem)
GSP (**G**eneral **S**imulation **P**rogram, Mark II)
MILITRAN
MONTECODE (**Monte** Carlo **Code**d Simulations)
OPS-3
SEAL (**S**imulation, **E**valuation and **A**nalysis **L**anguage)
SIMPAC
SIMSCRIPT
SIMTRAN
SIMULA
SOL
SOLPASS
SPS-1

All sorts of attempts have been made to "rank" these languages, or to try to advise a reader that one language would provide more capability than another. Other papers have been written which provided detailed comparisons of several of the languages. Presumably, in the latter case, the reader was to be in a better position to make a choice. In the opinion of this writer, all of such effort is futile. First, the choice of a language depends on an expert choosing among them, if all other factors are equal, which rarely occurs. Next, the choice depends upon the language in which the researcher can program. There is no virtue in a choice which cannot be utilized by the programmer, or at least, acquired by him within a reasonable time. Last, there is the question of compiler availability. Given a choice of one language over others, and given that the researcher is capable in that language, what earthly purpose is served if a compiler for that language is not available within a reasonable range of the intended user?

It has been the sad experience of the author, in which he is as much the culprit as the observer, that those who discuss several languages are rarely expert in more than one and never in more than two. However, with great pomposity, the "experts" discuss what each of several languages can, and cannot do, displaying, by their errors easily discernible by a real expert in any one language, that they have no real right to qualify or quantify that language. If several "single language" experts could objectively discuss any one paper which rates or ranks languages, it is forecast that much air could be released from the balloon.

To view these languages in retrospect has, in most cases, a nostalgic charm. It is even more important to now identify those which are still "active". With all due respect to second generation computers that are living a graceful old age in obscure universities, only those languages with compilers available on third generation machines are listed.

Arranged by families, the current list is:

GASP Family	*GPSS Family*	*SIMSCRIPT Family*
GASP	BOSS	CSL
HOCUS	EGPS	SEAL
PROSIM	FLOW SIMULATOR	SIMSCRIPT
SIMON	GESIM	SIMULA
SIMQUEUE	GPS	SOL
SPURT	GPS K	
UNS	GPSS	
	GPSS/NORDEN	

With a backward glance over our shoulder then, we have every right to look to the future with confidence. Any discipline which can invent at this prolific rate cannot but move forward with equal ability to formulate tools for the future work.

REFERENCES

1 R P BENNETT P R COOKEY S W HONEY C A KRIBBS M R LACKNER
SIMPAC user's manual, TM-602/000/00
Systems Development Corp April 1962

2 J J CLANCY M S FINEBERG
Digital simulation languages, A critique and a guide
Proceedings, Fall Joint Computer Conference 1965

3 O J DAHL K NYGAARD
SIMULA: A language for programming and description of discrete event system
Users Manual Norwegian Computer Center 1965

4 D G EBELING
A general purpose conversational time sharing program for probabilistic analysis
August 1969

5 J R EMSHOFF R L SISSON
Design and use of computer simulation models
Macmillan 1970

6 G EHRLING
PROSIM program system for simulation
DATASAAB-6361E 72.06.23 SAAB AKTIEBOLAG
Sweden 1966

7 E FAMOLARI
FORSIM IV, FORTRAN IV simulation language, user's guide
The Mitre Corporation SR-99 February 1964

8 *Flow simulator reference manual 70-00-617*
RCA April 1969

9 *General purpose simulation system/360 OS version 2, users manual*
IBM Corp SH20-0694-0 1969

10 *GESIM user's manual, GES-1022*

11 M GREENBERGER M M JONES J H MORRIS D N NESS
On line computation and simulation: The OPS-3 system
The MIT Press 1965

12 *Honeywell reference manual Order No 773 General purpose simulator K*
April 1969

13 *ICL reference manual 3386 1900 CSL*
October 1966

14 *ICL reference manual 4138 Simulation language SIMON*
January 1969

15 J KATZKE J REITMAN
Norden report 4269R0003, user's guide to conversational GPSS
December 1969

16 P J KIVIAT A A B PRITSKER
Simulation with GASP II
Prentice Hall 1969

17 P J KIVIAT R VILLANUEVA H M MARKOWITZ
The SIMSCRIPT II programming language R-360-PR
The RAND Corp October 1968

18 H S KRASNOW R A MERIKALLIO
The past, present and future of general simulation languages TM 17-7004
IBM Advanced Systems Development Division 1963

19 H M MARKOWITZ B HAUSNER H W KARR
SIMSCRIPT A simulation programming language
Prentice Hall 1963

20 *Militran reference manual AD 601-794*
Systems Research Group Inc

21 T H NAYLOR
Bibliography on simulation and gaming
TIMS College on Simulation and Gaming June 1968

22 A A B PRITSKER
JASP: A simulation language for a time shared system
RAND RM-6279-PR June 1970

23 *Proceedings of the* Second *Annual* Simulation *Symposium*
Tampa Fla January 1969

24 *Proceedings of the* Third *Annual* Simulation *Symposium*
Tampa Fla January 1970

25 *Proceedings of the* Second *Conference on* Applications *of* Simulation
New York 1968

26 *Proceedings of the* Third *Conference on* Applications *of* Simulation
Los Angeles 1969

27 *SIMTRAN reference manual*
IBM SDD Dept B85 July 1965

28 *Simulation,* Evaluation *and* Analysis *Language* (*SEAL*) *system reference manual 360D 15.1.005*
January 1968

29 D TEICHROEW J F LUBIN
Computer simulation—discussion of the technique and comparison of languages
Communications of ACM Vol 9 No 10 October 1966

30 D TEICHROEW J F LUBIN T D TRUITT
Discussion of computer simulation techniques and comparison of languages
Simulation October 1967

31 F M TONGE P KELLER A NEWELL
QUIKSCRIPT, A SIMSCRIPT-like language for the G-20
Communications of the ACM Vol 8 Number 6 June 1965

32 *UNS—Network simulator programmer's reference UP 7548*
1967

33 G A WICKLUND
SIMQUEUE—A queuing simulation model
College of Business Administration University of Iowa
Working Paper Series 69-13 July 1969

34 *Xerox General Purpose Discrete Simulator (GPDS)*
Sigma 5-9 computers Reference manual 90 17 58A
Xerox Data Systems April 1971

APPENDIX I

1. *Advertising*
   A. Media selection
   B. Impact on sales
   C. Campaign strategy evaluation
   D. Sales contests
   E. Customer contests
2. *Airlines*
   A. Runway utilization
   B. Terminal facility planning
   C. Crew scheduling
   D. Reservation system modeling
   E. Airliner seating and freight configurations
   F. Spare parts inventory
   G. Scheduling of service and maintenance facilities
   H. Time tables
   I. "Stack" modeling
   J. Cargo handling
   K. New airport facilities and locations
3. *Banking*
   A. Operation of bank floor (teller model)
   B. Behavior of on-line system
   C. Check transit model:
      1. Collection routes
      2. Centralized vs. decentralized processing
      3. Schedule arrival of inscriber operators
      4. Breakpoint for specially handled items
   D. Interest rate and other policy studies
   E. Special service studies (credit card, payroll, etc.)
4. *City Planning and Urban Renewal*
   A. Transportation networks
   B. Planning for services, facilities, etc.
   C. Welfare studies
   D. Crime and law enforcement studies
   E. Budget planning
   F. Information systems and record planning
5. *Communications*
   A. Information flow in networks
   B. Adaptive routing
   C. Study polling disciplines, concentrators, buffering, etc.
   D. Intercept and rating systems
   E. Evaluate telephone information system
   F. Maintenance and service facilities planning
   G. Studies of future growth needs
6. *Data Processing Systems*
   A. Response time, throughput analysis for real-time systems
   B. Organization of direct-access files
   C. Study queuing disciplines
   D. Storage allocation and buffering
   E. Study degraded performance
   F. Study operating system behavior
   G. Flow of jobs through a computer "shop"
   H. Priority assessments
   I. Equipment add-on effects
   J. Personnel and organization studies
7. *Distribution*
   A. Truck routing
   B. Optimization of distribution networks
   C. Number and location of warehouses
   D. Warehouse automation procedures
   E. Inventory management
   F. Schedule work crews
   G. Design truck docking facilities
   H. Design packaging facilities
   I. Information systems design
8. *Elevator Operation*
   A. Number of elevators
   B. Dispatching rules
9. *Enterprise Models*
   A. Flow of material, men, money, information. Evaluate all-over behavior—design for maximum profit. Policy studies at corporate levels
   B. Effect of reducing delays (such as installation of management information system)
   C. Personnel rules
   D. Advertising allocation
   E. Capital investment
   F. Diversification, merger and risk studies
10. *Gaming (other than war games)*
    A. Management games
    B. Purchasing games
    C. Creation of sporting events ("dream world series")
    D. Training models
11. *Insurance*
    A. Policy administration and accounting
    B. Investment studies
    C. Information and retrieval systems design

D. Statistical aging
E. New policy plans

12. *Job Shop Simulation*
A. Bottleneck elimination
B. Capacity-production studies (order backlog)
C. Work rules
D. Equipment evaluation and layout
E. Quality control

13. *Manufacturing*
A. Facilities planning
B. Assembly line balancing
C. Scheduling
D. Manpower allocation
E. Inventory management
F. Quality control
G. Information systems
H. Equipment maintenance
I. Subcontracting policies and schedules
J. Raw material acquisition
K. Plant location
L. Labor studies

14. *Marketing*
A. Pricing
B. Advertising
C. Sales force allocation
D. Product installation and acceptance
E. Competitive strategy
F. Product introduction studies
G. New product requirements

15. *Medical*
A. Behavior of real-time information system
B. Blood bank inventory
C. Admission/discharge policies
D. Hospital bed and patient scheduling
E. Scheduling of staff
F. Scheduling nurse activities

16. *Metal Processing*
A. Facility planning and scheduling
B. Warehousing
C. Ordering policies
D. Open pit mine design

17. *Monte Carlo simulation of complex deterministic problems*

18. *Paperwork Flow*

19. *Personnel Policies*
A. Hiring and promotion rules
B. Allocation, distribution, and movement of personnel

20. *Proposals*
Demonstrate new system feasibility to customer

21. *Railroads*
A. Yard operation
B. Network operations
C. Freight blocking strategies
D. Motive power assignment
E. Crew scheduling
F. Rapid transit—train scheduling
G. "Piggyback" studies
H. Equipment planning
I. Commuter rate studies

22. *Reliability*
A. Determine system availability
B. Spare parts requirements
C. Service crew and facility requirements
D. Economic batch size for quality
E. Failure rate studies
F. "Fail softly" systems—effect of duplexing

23. *Shipping*
A. Schedule port facilities
B. Schedule freighters, tugs, etc.
C. Cargo mix
D. Automation studies
E. "Fishybacks" and other competitive studies
F. Harbor design
G. Fleet composition
H. Labor practices

24. *Traffic Control*
A. Timing of traffic lights
B. Design turning rules at intersections
C. Test real-time control algorithms
D. Automated systems
E. Road planning
F. Safety studies

25. *Trucking*
A. Truck docking facilities
B. Scheduling
C. Routing and franchise studies
D. "Piggyback" and other competition
E. Rate studies
F. Information and retrieval systems

APPENDIX II

General Information about Digital-Discrete Simulation Languages

LANGUAGE NAME	TYPE	DEVELOPING ORGANIZATION	AUTHORS	COMPUTER IMPLEMENTATION	DOCUMENTATION OR REFERENCE
BOSS	Process	Burroughs	Roth, Meyerhoff & Troy	B5500, B6700	BOSS Applications Manual, Report #66099, Burroughs Corp., Paoli, Pa., July 1970
CLP	Process	Cornell University	Conway, Maxwell & Walker	CDC 1604	CPL Preliminary Manual, Dept. of Industrial Eng., Cornell U., Oct. 1963
CSL	Process	IBM(U.K.), Esso Ltd., ICL	Buxton & Laski	IBM 7090, 7094; 1900 Series ICL, ICL System 4	ICL Reference Manual 3386, 1900 CSL, Oct. 1966, and ICL Reference Manual 1105, CSL
EGPS	Process	Nippon Electric Co. Ltd.	Unk	NEAC Series 2200	NEAC Reference Manual, EGPS
ESP	Unk	Elliott Electric Co.	Unk	Elliott 503 & 803	Unk
FLOW SIMULATOR	Process	RCA CSD	Unk	RCA 3301 Spectra 70	Flow Simulator Reference Manual 70-00-617, April 1969
FORSIM IV	Event	Mitre Corporation	E. Famolari	IBM 7030	Mitre Progress Memorandum SR-99, FORSIM IV FORTRAN IV Simulation Language, User's Guide, E. Famolari, Feb. 1964
GASP	Event	U. S. Steel, Arizona State U.	Kiviat & Pritsker	With appropriate modifications, any computer with FORTRAN IV compiler	Simulation with GASP II, Kiviat & Pritsker, Prentice Hall, 1969
GEMS	Event	General Electric Co.	Markowitz	Unk	Unk
GESIM	Process	General Electric Co.	Unk	HIS Sys. 600 & 6000 Series	GESIM User's Manual GES-1022
GPDS	Process	Xerox Data Systems	Unk	Sigma 5-9	Xerox General Purpose Discrete Simulator, Sigma 5-9 Computers, Xerox Data Systems, April 1971
GPS	Process	Nippon Elec. Co. Ltd.	Unk	NEAC Series 2200	NEAC Reference Manual, GPS
GPS K	Process	Honeywell, Inc.	Unk	Series 200 (models 200/1200/1250/2200/4200	Honeywell Order No. 773, April 1969
GPSS	Process	IBM	Gordon and others	IBM 7090, 7094, 7040, 7044, System 360 UNIVAC 1107/1108/1110	General Purpose Simulation System/360 OS Version 2 Users Manual, SH20-0694-0, IBM Corp.
GPSS/360 NORDEN	Process	Norden Div. of United Aircraft Corp.	Katzke & Reitman	IBM System 360 w/2250 Display Unit	Norden Report 4269R0003, Users Guide to Conversational GPSS, December 1969

APPENDIX II (Cont'd)

LANGUAGE NAME	TYPE	DEVELOPING ORGANIZATION	AUTHORS	COMPUTER IMPLEMENTATION	DOCUMENTATION OR REFERENCE
GSP MK II	Unk	U. S. Steel Co. Ltd.	Unk	Ferranti Pegasus	Operational Research Report, #118/ORD 10/ TECH, 1964
HOCUS	Event	P-E Consulting Group Ltd.	R. Hills	Any with FORTRAN compiler	HOCUS Manuals I & II
JASP	Event	RAND Corp.	Pritsker	JOSS Language Computer (RAND only)	RAND Memo RM-6279-PR, June 1970
MILITRAN	Event	Office of Naval Research & Systems Research Group, Inc.	Unk	7090, 7094	MILITRAN Reference Manual, AD 601-794, Systems Research Group, Inc. (Clearinghouse)
MONTECODE	Unk	Unk	Kelley, D. H. & Buxton, J. N.	Unk	An Interpretive Program for Monte Carlo Simulations, Computer Journal, V, 1962
OPS-3	Event	M. I. T.	Greenberger, Jones, Morris & Ness	Only with M.I.T. Time Sharing CTSS System	On-Line Computation and Simulation OPS-3, Greenberger, et al, MIT Press, 1965
PA3	Unk	General Electric Co.	Ebeling and Hurst	GE Mk II Time Sharing	A General Purpose, Conversational Time-Sharing Program for Probabalistic Analysis (PA3-1969 version of PAL), August 1969
PROSIM	Unk	Data Saab	Unk	D21, D22	Unk
QUIKSCRIPT	Event	Carnegie Inst. of Technology	Tonge, Keller & Newell	G-20	Communications of the ACM, June 1965
QUIKSIM	Process	National Cash Register Co.	Weamer	NCR 315 RMC	Proceedings of the Third Conference on the Applications of Simulation, December 1969
SEAL	Event	IBM	Braddock & Dowling	IBM System 360	Simulation, Evaluation and Analysis Language (SEAL), IBM System Manual, 360D 15.1.005
SIMON	Process	Bristol College of Science & Technology and ICL	Unk	Elliott 503, 803, ICL 1900 Series	ICL Reference Manual 4138, Simulation Language SIMON, January 1969
SIMPAC	Process	Systems Development Corp.	Bennett, et al	IBM 7090, 7094	SIMPAC Users Manual, SDC TM602/000/00, Bennett et al, April 1962
SIMQUEUE	Event	University of Iowa	Wicklund	IBM System 360	Working Paper Series 69-13, College of Business Administration, University of Iowa

APPENDIX II (Cont'd)

LANGUAGE NAME	TYPE	DEVELOPING ORGANIZATION	AUTHORS	COMPUTER IMPLEMENTATION	DOCUMENTATION OR REFERENCE
SIMSCRIPT	Event	RAND Corp.	Markowitz, Hausner & Karr	IBM 7090, 7094 7040, 7044, 360 CDC 3600, 3800, 6400, 6600, 7600 UNIVAC 494, 1107, 1108, 1110, RCA Spectra 70 series NCR 200 series, PHILCO 2000 series, HIS 615, 625 635, 655, 6030, 6040, 6060, 6080, STANDARD IC-6000	SIMSCRIPT, A Simulation Programming Language, Markowitz, Hausner & Karr, Prentice Hall, 1963
SIMSCRIPT II	Event	RAND Corp.	Kiviat, Markowitz, Hausner & Villaneueva	IBM System 360, RCA Spectra 70 (object code only)	SIMSCRIPT II - A Programming Language, Kiviat, Villaneueva and Markowitz, Prentice Hall 1969
SIMTRAN	Event	IBM	Braddock, Dowling and Rochelson	IBM 7030	Reference Manual, IBM, SDD, Dept. B85, July 1965
SIMULA	Process	Norwegian Computer Center	Dahl & Nygaard	UNIVAC 1107, 1108 1110, CDC 6400, 6600, 6700, 7600 Burroughs B5500, B6000 series	SIMULA - A language for Programming and description of discrete event system, Users Manual, Dahl & Nygaard, Norwegian Computer Center, 1965
SOL	Process	Burroughs Corp. & Case Inst. of Tech	Knuth & McNeley	Burroughs B5000/ 5500, UNIVAC 1107 1108	SOL - A symbolic language for general purpose systems simulation, Knuth and McNeley, 1963
SOLPASS	Process	U.S. Army (Ft. Monmouth) and International Computer Sciences, Inc.	Armstrong, Ulfers, Miller & Page	Burroughs B5500	Unk
SPURT	Event	Northwestern U.	Mittman & Goldberg	CDC 3400, 6000 series, IBM	SPURT Users Manual, Vogelbach Computing Center, Northwestern U.
UNISIM	Process	Bell Telephone Labs.	L. A. Gimpelson & J. H. Weber	UNIVAC 1108	UNISIM - A Simulation Program for Communication Networks - Case 38931, Oct. 9, 1963
UNS	Process	UNIVAC	Unk	UNIVAC 490 series 1107, 1108, 1110	Network Simulator Programmer's Reference, UP 7548, 1967

OSSL—A specialized language for simulating computer systems*

by PREM BHUSHAN DEWAN

Mississippi State University
Starkville, Mississippi

and

C. E. DONAGHEY and JOE B. WYATT

University of Houston
Houston, Texas

INTRODUCTION

The need for evaluating the performance of contemporary computer systems is well recognized by the manufacturers as well as the users of these systems. The evaluation is difficult because of the complexities and sophistication of the computer hardware and software system design. The computer manufacturers have produced an abundance of literature encompassing somewhat subjective evaluations of their products. Unfortunately, relatively little effort has been directed by the manufacturer toward the development of generalizable scientific tools for the purpose of a quantified evaluation of the performance of computer systems operating in a specified environment. The need for such generalizable scientific tools for studying the behavior of a given computer system in a specified environment can hardly be over-emphasized.

Contemporary computer systems are, in general, architectured from organized semi-independent processing modules which share a finite set of modular resources. The performance characteristics of a computer system are partly governed by the characteristics of the various modules and partly by the interactions and interrelationships among these modules. A performance evaluation instrument for these systems should provide an insight into the dynamics of the interactions among the various components of the system. This hypothesis strongly suggests simulation techniques for investigating the behavior of computer systems.

Simulation, by definition, is a process of conducting experiments on a model representing an abstraction of the system. Simulation may take place only if a model of the system under investigation is available.

As previously hypothesized, computer systems may be represented in terms of components, variables, parameters in conjunction with algorithms which describe corresponding logical and functional relationships. However, the dynamic complexity of these systems coupled with the desired level of detail, does not facilitate the translation of these models into a set of equations which are analytically tractable. Nonetheless, it is possible to apply the description of the computer system (model) to a given set of input to the system. The manner in which the model reacts to this input may be summarized in terms of the variables of interest such as the time consumed by the management functions and the fraction of the time the various system components are busy. By varying the input and/or varying the model, an investigator can observe the variables of interest and make appropriate decisions. The enormity of computations and the volume of data involved, in investigating systems in this manner, can be handled effectively if the model is translated into a computer program. The validity of this approach is evidenced by the simulation studies conducted by Nielsen,[16,17] Katz,[6] Scherr,[22] and Rehman and Gangware[19] among others. These simulation studies were reported to be successful. The building of simulation models requires specialized skills and time. These requirements act as deterrents for conducting generalized simulation studies of computer systems.

There is a need in several areas for a mechanism that enables the construction of working simulation models (at several levels of detail) in relatively short periods

* Some of the work leading to this paper was supported in part by ONR-THEMIS Contract N00014-68-A-0151

of time. Such a mechanism would not only be useful to the designers and users of the computer systems as a "quick-look" mechanism but would also have a significant pedagogical value. Students of operating systems could gain in-depth knowledge through designing and observing operating systems by the use of such a mechanism.

In the development of the OSSL language and simulator an effort has been made to provide the user with the capability to represent all components of a computer system including memories, processors, and input-output systems in both an associative and a hierarchical relationship. The OSSL language has not been written for usage in areas other than computer systems simulation and is intended to be limited to this area of application. The known limitations of the language lie in the representation of detailed discrete phenomena such as a particular memory mapping algorithm, a hybrid input-output algorithm, etc., although most, if not all, of these cases can be represented stochastically in considerable detail. A similar effort has been made in OSSL to enable the representation of software processors and task sequences, stochastically, in sufficient detail and accuracy to provide for the simulation of any specific computer system work load.

THE SIMULATION LANGUAGE

This paper presents an overview of a language developed for structuring simulation models of computer systems. A more detailed description of the OSSL Language and its use is presented in a User's Guide.[25] The various language elements reflect the phraseology in use by those involved in the implementation of computer systems. The format-free language has been implemented in a subset of the FORTRAN IV language for the following reasons:

(a) FORTRAN IV is available on almost all medium- to large-scale computer systems. Special features of the language were purposely avoided in the coding process to facilitate usability on a variety of implemented systems.
(b) FORTRAN IV has the potential to accommodate the flow oriented structure of GPSS, and also the entity, attribute, set, state, and event concepts of SIMSCRIPT.
(c) FORTRAN IV provides the facility to construct adequate specialized data structures needed for simulating computer systems.
(d) Implementation of the language interpreter is much easier in a general purpose language like FORTRAN, than in the special purpose simulation languages like GPSS, SIMSCRIPT, SIMULA, etc.
(e) The simulator and the language interface have been coded in FORTRAN IV in order to provide user modularity. The various OSSL segments have been represented via FORTRAN subroutines which are as accessible as provided for by the particular FORTRAN compiler.

A simulation model for a given computer system consists of components representing

(a) the hardware characteristics and system configuration,
(b) the operational philosophy of the system, and
(c) the environment in which the system is to function.

The simulator employs the asynchronous timing technique for advancing the model through the simulated time. This implies that the state of the system is updated upon the occurrence of events. An event may be the start of a new job, initiation/completion of data transmission, interruption of a running program to provide CPU service relative to a different user job, etc. All the processes to be simulated are viewed as a series of events. The updating of the state of the system consists in changing the simulation parameters as well as the scheduling of future events induced by the event causing the update. All future events thus scheduled are filed in a threaded list. The events are retrieved from the list in the order of their scheduled occurrence of time for updating the state of the system. Such a scheme positions events on the time line and the updating mechanism relates the events to their respective processes, thereby enabling the handling of concurrent operation of the various processes.

The language described herein allows for a modular construction of simulation models representing these components. Each of the components may consist of one or more segments provided by the language. The three components of the model may be structured as follows:

HARDWARE CHARACTERISTICS AND SYSTEM CONFIGURATION

This component may be defined through the following two segments of the language:

Central processing units (CPUs)

The CPUs are considered to be interruptible entities capable of executing instructions related to a single request at any instant of time. All requests for CPU service have an associated priority level. The language allows for the definition of up to 15 CPUs in the system.

```
              CPU DATA
CPU  1
     QUEUE 7
     UNINTERRUPTIBLE LEVEL 45
     TIME QUANTUM .050
     INSTRUCTION TIMES .00001 .000015 .00005
     TABULATE BUSY TABL
        END OF CPU DATA
```

Figure 1

Figure 1 shows an example of the CPU definition segment. CPU numbered 1 is being defined. Requests for CPU service are entered in the queue number 7 if they cannot be met immediately.

The CPU goes into an uninterruptible state if the request being executed has a priority level of 45 or higher. The time quantum is specified to be .05 units of time. The CPU can execute one instruction each of class 1, 2 and 3 in .00001, .000015 and .00005 units of time respectively. The language allows for up to 10 classes of instructions. The time required for instructions belonging to classes 3-10 is specified to be zero. The continuously busy spans of the CPU are to be tabulated in the frequency table labelled TABL.

Devices and channels

Characteristics of the peripheral devices like the card-readers, card-punch equipment, teletypes, CRT units, etc., are defined in this segment. The system configuration is implied through the definition of each device. Multiplexor channels are not to be included since these are transparent to the simulation model. This is because a multiplexor channel is always available to the device controllers attached to it.

Figure 2 shows a sample definition of the segment. The system configuration implied is shown in figure 3. The characteristics of the CPU would be defined in the CPU segment.

The segment, as defined, specifies one card-reader, one line printer, one card-punch and two discs. The strings TCR, TCP, T1, T2, T3 and T4 are the labels assigned to the user defined distribution functions. Requests for the selector channel are to be entered in queue 5 if they cannot be attended to immediately. Similarly, requests for disc number 2 are to be entered in queue number 3 if they cannot be attended to immediately.

THE OPERATIONAL PHILOSOPHY

This component of the simulation model embodies the manner in which the user jobs flow through the system. The flow of jobs involves movement of data among the system components, allocation of primary memory, scheduling of jobs for execution and file management functions. The manner in which these

```
    DEVICES AND CHANNELS SEGMENT
TOTAL    CR       1
TOTAL    LP       1
TOTAL    DISCS    2
TOTAL    CP       1
    DEVICE      CR 1
                TRANSFER TIME TCR
    DEVICE      CP 1
                TRANSFER TIME TCP
    DEVICE      LP 1
                TRANSFER TIME T1
    DEVICE      DISC 1
                SEEK TIME T2
                TRANSFER TIME T3
                SELECTOR CHANNEL 1
    DEVICE      DISC 2
                TRANSFER TIME T3
                SEEK TIME T4
                SELECTOR CHANNEL 1
                QUEUE 3
    CHANNEL     1
                QUEUE 5
        END OF DEVICES AND CHANNEL DEFINITION
```

Figure 2

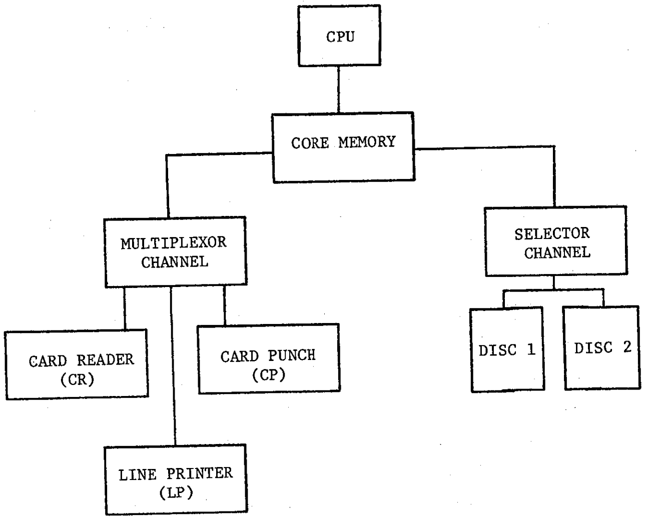


Figure 3

activities are to be performed are defined by the following segments:

Bufferpools

The language allows for the operation of up to twenty bufferpools numbered 1 to 20. A request for space in the bufferpool is disposed of in the following manner:

(a) User defined criterion are applied to the request to determine if the request is to be met.
(b) Number of instructions (say n) to be executed for applying the criterion are generated in accordance with the user specified distribution function.
(c) If the request is to be met, the number of instructions to be executed (say m) for the assignment of space are generated in accordance with the user specified distribution function.
(d) If the request is not to be met immediately, the number of instructions (say m) to be executed to enter a specified queue are generated in accordance with the user specified distribution function.
(e) A CPU from the specified list is selected to execute $n+m$ instructions.
(f) The request is either met or placed in a queue upon execution of $n+m$ instructions.

Figure 4 shows a sample definition of the bufferpool segment. Bufferpool numbered as 1 containing 100 buffers of a unit space each is being defined. The number of instructions for decision, assign, refusal and release activities follow the user defined distribution functions labelled DISA, DISB, DISC, DISD. Requests for bufferspace are to be placed in queue number 10 if they cannot be met. MXA and MB are the labels assigned to the proportion of the various classes of instructions to be executed by the CPU number 1. The priority level for the CPU service associated with the assignment and release of bufferspace has been specified as 92.

```
          BUFFERPOOL DEFINITION
BUFFERPOOL 1
    FIXED SIZED BUFFERS
    NUMBER OF BUFFERS 100
    SPACE 100
    INSTRUCTIONS FOR DECISION DISA MIX MXA
    INSTRUCTIONS FOR ASSIGN DISC MIX MXA
    INSTRUCTIONS FOR REFUSAL DISC MIX MXA
    INSTRUCTIONS FOR RELEASE DISD MIX MB
    CPU 1
    INTERRUPT LEVEL FOR ASSIGN 92
    INTERRUPT LEVEL FOR RELEASE 92
DECISION RULES FOR BUFFER SPACE
  ASSIGNMENT
    FILE IS USING LESS THAN 3 BUFFERS OR
      AT LEAST 1 BUFFER AVAILABLE FOR EACH FILE
     END OF BUFFERPOOL DEFINITION
```

Figure 4

The request for bufferspace is to be met if the file requesting space is using less than 3 buffers or at least 1 buffer is available for each file attached to the bufferpool.

Movement of data

Movement of data in the system is described via the definition of one or more PROCEDURES. An example of a procedure is the set of activities associated with the input of a job in a batch-processing system like the XDS system SIGMA 7 symbiont and the IBM System/360 SPOOL (DOS/360 POWER and OS/360 HASP).

The input procedure may consist in the movement of the source statements from a card reader on to the disc. This movement of data would involve various activities like the request and assignment of bufferspace, activation of the card-reader, deposition of the card image into a buffer, emptying of the buffer onto the disc, release of the bufferspace, and update of the various tables.

The various tasks being performed by a computer system under the direct control of its operating system may be grouped into procedures. These procedures are activated by the services requested by a user's job. A procedure consists of an ordered list of statements and thus are flow oriented. The language provides 38 different statements for defining procedures. There are statements to request system services like the movement of data, request/release of bufferspace, request for CPU service via an interrupt, etc. Statements have also been provided for the management of records in files and interrogation of the attributes of files. The language also provides statements for controlling the flow within a procedure. The procedures defined by the user are assigned unique labels. Individual statements comprising the procedure may be assigned a label. Some examples of the statements available are as follows:

Statement:

MOVE TO DISC 2

Execution of this statement causes data to be moved to Disc 2.

Statement:

INTERRUPT CPU 1 PRIORITY 95 QUANTITY A MIX MX

This statement generates a random value, say *n*, from the user defined distribution function labelled A. A request with a priority level of 95 is placed on CPU 1 to execute *n* instructions of the mix labelled as MX.

Statement:

REQUEST 1 FROM BUFFERPOOL 2

A request for space from the bufferpool number 2 is generated and disposed of as prescribed by the user in the BUFFERPOOL segment.

Statement:

ASSIGN FILE 6

The statement creates file 6 for the job being handled by the procedure.

Statement:

ATTACH FILE 8 TO BUFFERPOOL 1

This statement attaches file 8 belonging to the job to the bufferpool 1.

Statement:

CHECK IF RECORD READY

The statement gets executed only if a record is available for reading.

Statement:

CHECK IF RECORD IN FILE , ET

The flow is transferred to the statement labelled ET if the file is empty, otherwise the flow reaches the next sequential statement.

Statement:

INCREMENT DATA IN FILE 6 BY 2

This statement increases the number of records in file 6 by 2.

Statement:

REQUEST CHANNEL 1 2

This statement is considered executed only when either the selector channel 1 or the selector channel 2 is made available to the procedure.

A sample procedure to mode input from a teletype into the system is shown in Figure 5.

```
PROCEDURE IN
    REQUEST 1 FROM BUFFERPOOL 3
    MOVE FROM TELETYPE 10
    INTERRUPT CPU 1 PRIORITY 90 QUANTITY A MIX X
TERMINATE
```

Figure 5

```
        SCHEDULER DEFINITION
QUEUE 7
CPU 1
JOBS ARE SLICED
MAXIMUM JOBS UNDER PAGING 5
  END OF SCHEDULER DEFINITION
```

Figure 6

Job scheduler

Assignment of CPU for executing the code specified by the user's job is handled by the job scheduler. The model provides for the simulation of batch-processing, time-sharing and multiprogramming computer systems by an appropriate definition of scheduler. Figure 6 shows a sample definition of the scheduler. A time-slicing system is specified. Jobs awaiting CPU assignment for the execution of code are placed in queue number 7. CPU number 1 is to be used for executing the code. The number of jobs under paging at any time may not exceed 5.

Memory management

This segment is for the purpose of defining the various parameters as well as the decision rules to be used for allocating primary memory to the various jobs. A sample definition of the memory management functions is shown in Figure 7. The primary memory consists of a single partition containing 512 pages. All the user specified code is relocatable. Pages not in use are used for meeting page demands. If no such page is available, a set of pages is generated. The pages belonging to this set have been accessed at least once and do not belong to the job demanding the page. The page that has been in residence the longest among the members of the generated set is selected for meeting the demand. A page is swapped only if it has been modified. The rolling-in and rolling-out of pages are in accordance with the user defined procedures labelled RIN and ROUT respectively.

```
    MEMORY MANAGEMENT DEFINITION
PARTITION 1
    LAST PAGE OF THE PARTITION 512
RELOCATABLE CODE
GENERATE A SET OF PAGES AS FOLLOWS:
    PAGE DOES NOT BELONG TO THE JOB REQU ESTING PAGE
    PAGE HAS BEEN ACCESSED AT LEAST ONCE
PAGE ASSIGNED FIFO
ROLL IN PROCEDURE RIN
ROLL OUT PROCEDURE ROUT
PAGES SWAPPED ONLY IF MODIFIED
    END OF MEMORY MANAGEMENT
      DEFINITION
```

Figure 7

File management

CPU service, if any, in regard to the opening, closing, attaching/unattaching of files during the execution of procedures, is specified in this segment.

Figure 8 shows a sample definition of this segment. The specifications imply that a request for CPU service is generated if the files are opened, closed, and attached/unattached. The number of instructions to be executed by the CPU number 1 follow the distribution function labelled INST. The priority level associated with the request for CPU service is specified to be 97.

```
              FILE MANAGEMENT
CPU 1
INSTRUCTIONS FOR OPENING FILES INST MIX S2
INSTRUCTIONS FOR CLOSING FILES INST MIX S2
INSTRUCTIONS FOR ATTACH/UNATTACH FILES INST MIX S2
INTERRUPT LEVEL FOR OPENING FILES 97
INTERRUPT LEVEL FOR CLOSING FILES 97
INTERRUPT LEVEL FOR ATTACH/UNATTACH FILES 97
END OF FILE MANAGEMENT DEFINITION
```

Figure 8

THE ENVIRONMENT

This component of the simulation model embodies the services that the system is called upon to perform. The language provides for structuring the environment in terms of jobs. Since the jobs require a variety of system services, the language allows for the classification of jobs into numbered types. All the jobs belonging to a particular type demand the same system services in the same order. However, the amount of service demanded by jobs of the same type may be different. The services demanded by the job may be classified into three categories, viz., entry of the job into the system, computations as specified by the job and the delivery of the computational results by the system. Entry of the job, the delivery of the computational results and the manner in which the computations are executed is fixed by the design of the system (as represented via the hardware configuration and the operational philosophy) and is described by specifying job steps relative to each type of job.

Code

The computations to be performed are represented by the code to be executed. The code may pertain to the compiling, loading, and the job itself. The language provides the facilities to specify the code to be executed. The language allows for the structuring of the various software processors from the user defined code.

The software representation is at a macroscopic level. A software program to be represented is divided into one or more "blocks." Each of these blocks contains the instructions to be executed, the specific pages needed in core for the execution of the block. In addition, the block may also contain specifications on the pages modified and the frequency with which the various pages are accessed during the execution of the code contained in the block.

A block is assigned a unique label so that a reference may be made to it for the purposes of structuring software processors, specification of code for the user's jobs and transfer of control during execution.

Figure 9 shows a sample definition of the code. Three blocks of code labeled EX, ST, AND EX2 have been defined. A brief explanation of the code contained in the block labelled EX is as follows:

Pages 1, 3, 7, and 9, relative to the job are needed in core before the code may be executed. Pages 3 and 7 will be modified as a result of the execution of the code.

The statement

INITIATE READ 1 IC 3

implies that the procedure associated with the type 1 read is to be invoked. The number of records to be read follow the distribution labelled IC and the file number 3 is involved in the read statement. Control is transferred to the next statement without awaiting the completion of the read.

The statement

E READ 2 RDZ 5 ST

specifies that records from file 5 are to be read from file 5 in a manner defined by the procedure associated with the type 2 read. The number of records to be read is to be randomly generated in accordance with the distribution function labelled RD2. In case file 5 is empty, control is transferred to the block labelled ST. Control is transferred to the next statement only after the read statement has been completed. E is the label assigned to this statement.

The statement

COMPUTE COM MX

specifies that the CPU is to be requested to execute instructions of a mix defined by the label MX. The number of instructions to be executed are randomly generated from the distribution function labelled COM.

The statement

WRITE 2 WT1 7

specifies that a number of records, randomly generated from the distribution function labelled WT1, be written into file 7. The manner in which the records are to be written into file 7 is specified via the procedure associated with type 2 write. Control is transferred to the next statement only after the write operation has been completed.

The statement

```
LOOP LE E
```

specifies that the statements starting with the one labelled E and ending with the one before the LOOP statement are to be executed a number of times randomly generated from the distribution function labelled LE.

```
         CODE DEFINITION
EX       BLOCK
         PAGES NEEDED 1 3 7 9
         PAGES MODIFIED 3 7
         INITIATE READ 1 IC 3
E        READ 2 RD2 5 ST
         COMPUTE COM MX
         WRITE 2 WT1 7
         LOOP LE E
         TRANSFER .4 EX 2
         WRITE 2 WT2 7
ST       BLOCK
         STOP
EX2      BLOCK
         PAGES NEEDED 2 4 5 6 8 9 10
         PAGES MODIFIED 4 8
E        READ 2 RD 6 ST
         COMPUTE CA MP
         WRITE 4 WT 8
         JUMP .8 E
         STOP
      END OF CODE DEFINITION
```

Figure 9

The statement

```
TRANSFER .4 EX2
```

would result in the generation of a random value from the uniform distribution over the interval (0., 1.). If the value of the number generated is less than or equal to .4, the control is transferred to the block labelled EX2, otherwise the control reaches the next sequential statement.

Input/output

The procedures associated with the different types of reads/writes are declared as shown in Figure 10. RDA, RDB, AD, W, AND RITE are the labels of the user defined procedures.

```
     INPUT/OUTPUT PROCEDURES
      READ 1  PROCEDURE RDA
      READ 2  PROCEDURE RDB
      READ 3  PROCEDURE AD
      WRITE 1 PROCEDURE W
      WRITE 2 PROCEDURE RITE
  END OF INPUT/OUTPUT DEFINITION
```

Figure 10

Software processors

Software processors may be defined from the specified code. A sample definition of a software processor named FORT consisting of 31 pages is shown in Figure 11. EXE is the label of a block of code.

```
      SOFTWARE PROCESSOR
       PROCESSOR FORT
          STARTS AT EXE PAGES 31
  END OF SOFTWARE PROCESSOR
           DEFINITION
```

Figure 11

Code specifications for the jobs

The number of pages and the code of a job arriving in the system is specified in the job list segment. Each entry in the list contains the number of pages and the reference to a label of a block of code. The job list is subdivided. Each sub-list belongs to a different job type. The sub-lists are circular and hence endless. Each of the sub-lists has a pointer indicating the entry to be used for assigning the number of pages and the code to a job arriving in the system. The pointer moves to the next entry after the current entry has been used. Figure 12 shows a sample job list.

```
                 JOB LIST
Type 1
   PAGES 143  STARTS AT BLOCK C1
   PAGES  78  STARTS AT BLOCK LA
Type 2
   PAGES  47  STARTS AT BLOCK LB
   PAGES  43  STARTS AT BLOCK CN
          END OF JOB LIST
```

Figure 12

Job description

The job type is characterized by the specification of the relationship between the various procedures and processors. A job type is specified in terms of steps to be executed sequentially. Each step consists in either the execution of a procedure or the execution of code. Figure 13 shows a sample job description.

```
    JOB DESCRIPTION DEFINITION

  Type 1

     JOB STEPS ARE AS FOLLOWS:
         PROCEDURE IN FILE 7 DEVICE 2
         EXECUTE FO
         EXECUTE LO
         EXECUTE JOB
         PROCEDURE OUT FILE 8
       END OF JOB DESCRIPTION
```

Figure 13

IN and OUT are the labels of the user defined procedures, whereas FO and LO are the labels of the user defined software processors.

The language embodies other elements for defining:

(a) the various distribution functions,
(b) the various frequency tables,
(c) the various cumulative distribution functions for generating the job types,
(d) queue disciplines,
(e) initial conditions,
(f) limits on the number of jobs in the system at any instant of time, and
(g) other simulation parameters like the seed for generating random numbers, length of a simulation run and the number of reports to be generated.

USE OF THE MODEL

Several simulation studies have been carried out to check the operation of the generalized model. One such case is included to serve as an example for demonstration of some of the uses of the generalized model. The particular simulation study concerns an on-line enquiry system as described on page 79 of the *General Purpose Simulation System*/360: *Introductory User's Manual*, IBM Inc., Form GH20-0304-4. A GPSS construct of the enquiry system appears on pages 82-83 of the referenced material. The on-line enquiry system was selected for simulation due to its relative simplicity and because of the availability of published results for comparison purposes.

The experiment consisted of processing the OSSL version of the basic GPSS model in order to verify the similarity of the results. The OSSL model was constructed using as nearly as possible identical parametric information to that assumed for the GPSS representation of the five teletype computer enquiry system. The basic OSSL model is shown in Figures 14a, 14b, and 14c.

An explanation of various statement types may be found in Reference number 25 (Wyatt & Dewan, 1971).

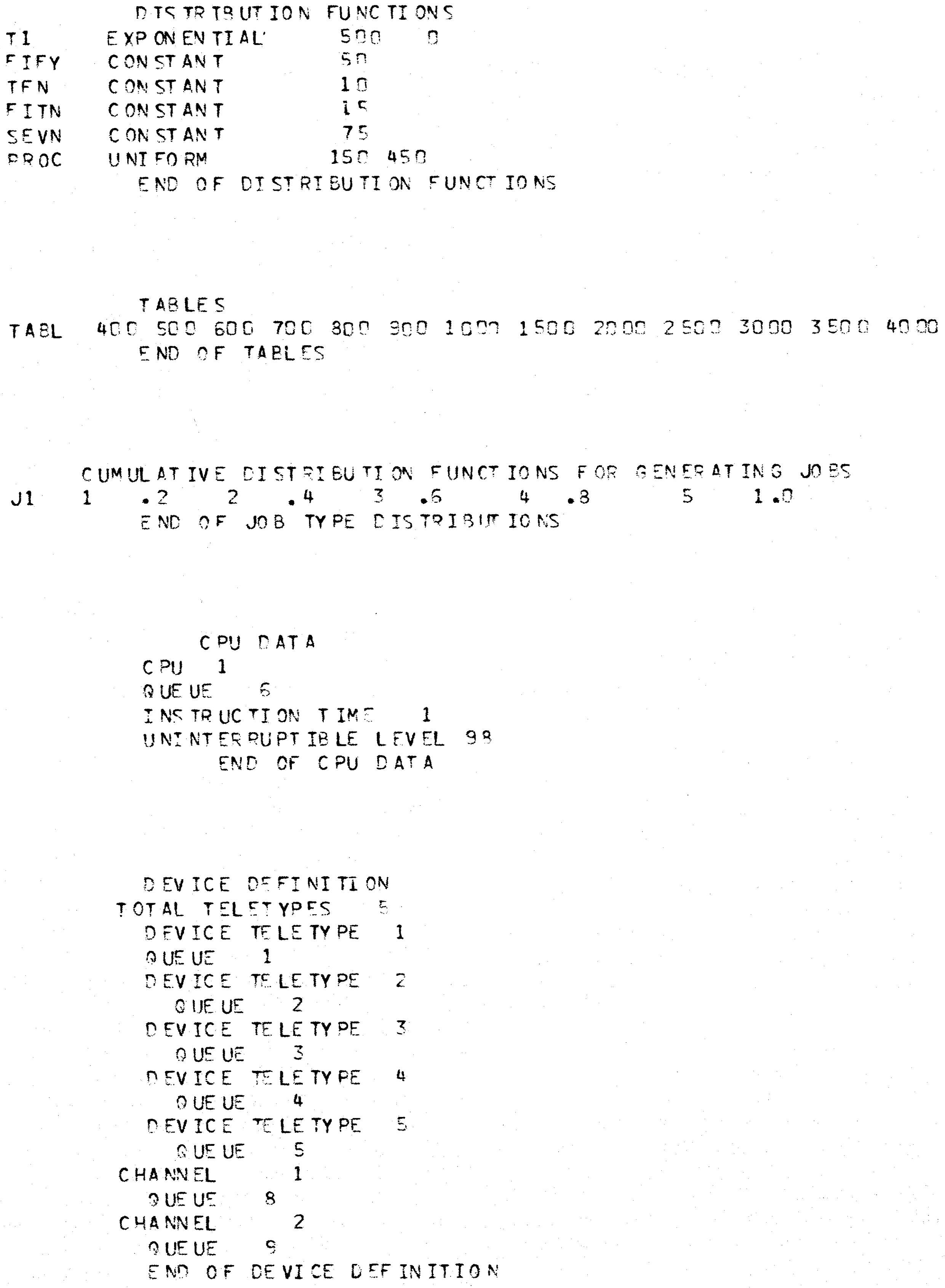
All the jobs arriving in the system require the same set of services from the system. The only distinction between jobs is the teletype number from which they originate. This distinction was used to build the five types of jobs that the model services. Jobs belonging to type *n* implies that they originated from the teletype number *n*. Thus a job type 3 implies that the job originates from the teletype number 3.

The job description segment defines the job steps associated with each of the five job types. Each of the five job types consists of a single step, viz., execution of the procedure labelled ENQY. The specifications imply that a job arriving in the system would cause the activation of the procedure ENQY relative to this job. The file associated with the procedure will be 0. The device number associated with ENQY would depend on the type of the job. Thus if the arriving job is of type 2 the device number 2 would be associated with ENQY. The priority associated with jobs is set to 101 due to lack of specifications.

The Bufferpool Definition segment specifies that the system is to contain a bufferpool numbered as 1. All the buffers in this pool are of a fixed size. The total space in the pool is 10 units divided amongst 10 buffers. Requests for bufferspace are to be placed in queue 7 if they cannot be satisfied immediately. Number of instructions to be executed by the CPU for the assignment and release of bufferspace is specified as zero (by default). Lack of decision rules specifications for the assignment of bufferspace imply that the requests for bufferspace are to be met if bufferspace is available.

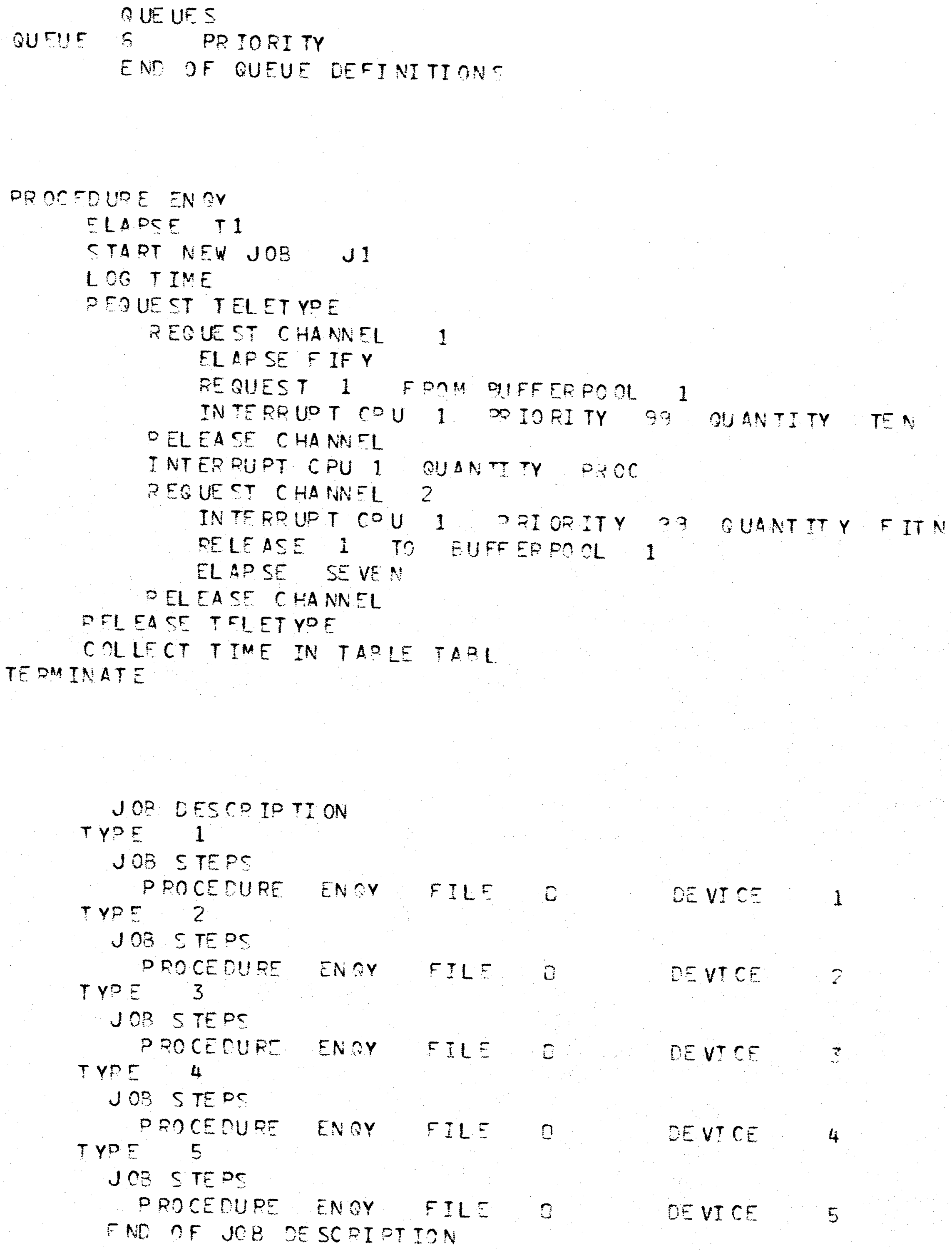
The Initial Condition segment specifies that at simulated time zero one job is to arrive in the system. The type of this job is to be determined by the cumulative distribution function labelled J1.

The Simulated Time Specification segment specifies that the model is to be operated for 1020000 units of simulated time after which a report is to be generated. Lack of the random number seed specification implies that the system is to use the built-in random number seed during the simulation process.



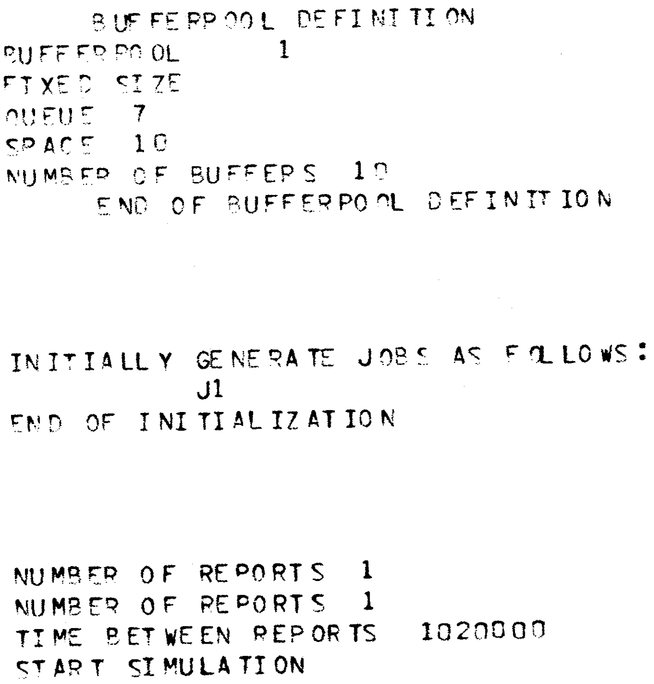
```
          DISTRIBUTION FUNCTIONS
TI      EXPONENTIAL       500    0
FIFY    CONSTANT          50
TEN     CONSTANT          10
FITN    CONSTANT          15
SEVN    CONSTANT          75
PROC    UNIFORM          150  450
          END OF DISTRIBUTION FUNCTIONS



          TABLES
TABL    400 500 600 700 800 900 1000 1500 2000 2500 3000 3500 4000
          END OF TABLES



        CUMULATIVE DISTRIBUTION FUNCTIONS FOR GENERATING JOBS
J1      1    .2    2    .4    3   .6     4   .8       5    1.0
          END OF JOB TYPE DISTRIBUTIONS



              CPU DATA
          CPU   1
          QUEUE    6
          INSTRUCTION TIME     1
          UNINTERRUPTIBLE LEVEL  98
               END OF CPU DATA



          DEVICE DEFINITION
        TOTAL TELETYPES     5
          DEVICE TELETYPE   1
          QUEUE    1
          DEVICE TELETYPE   2
            QUEUE    2
          DEVICE TELETYPE   3
            QUEUE    3
          DEVICE TELETYPE   4
            QUEUE    4
          DEVICE TELETYPE   5
            QUEUE    5
        CHANNEL       1
          QUEUE    8
        CHANNEL       2
          QUEUE    9
          END OF DEVICE DEFINITION
```

Figure 14a—OSSL model



```
       QUEUES
QUEUE  6     PRIORITY
       END OF QUEUE DEFINITIONS



PROCEDURE ENQY
     ELAPSE  T1
     START NEW JOB   J1
     LOG TIME
     REQUEST TELETYPE
         REQUEST CHANNEL    1
            ELAPSE FIFY
            REQUEST  1    FROM BUFFERPOOL   1
            INTERRUPT CPU  1   PRIORITY  99  QUANTITY  TEN
         RELEASE CHANNEL
         INTERRUPT CPU 1   QUANTITY  PROC
         REQUEST CHANNEL  2
            INTERRUPT CPU  1   PRIORITY  99  QUANTITY  FITN
            RELEASE  1  TO  BUFFERPOOL  1
            ELAPSE   SEVEN
         RELEASE CHANNEL
     RELEASE TELETYPE
     COLLECT TIME IN TABLE TABL
TERMINATE



       JOB DESCRIPTION
    TYPE    1
       JOB STEPS
         PROCEDURE  ENQY   FILE   0       DEVICE     1
    TYPE    2
       JOB STEPS
         PROCEDURE  ENQY   FILE   0       DEVICE     2
    TYPE    3
       JOB STEPS
         PROCEDURE  ENQY   FILE   0       DEVICE     3
    TYPE    4
       JOB STEPS
         PROCEDURE  ENQY   FILE   0       DEVICE     4
    TYPE    5
       JOB STEPS
         PROCEDURE  ENQY   FILE   0       DEVICE     5
      END OF JOB DESCRIPTION
```

Figure 14b—OSSL model



```
      BUFFERPOOL DEFINITION
BUFFERPOOL       1
FIXED SIZE
QUEUE    7
SPACE    10
NUMBER OF BUFFERS  10
      END OF BUFFERPOOL DEFINITION



INITIALLY GENERATE JOBS AS FOLLOWS:
            J1
END OF INITIALIZATION



NUMBER OF REPORTS  1
NUMBER OF REPORTS  1
TIME BETWEEN REPORTS    1020000
START SIMULATION
```

Figure 14c—OSSL model

The output generated by the generalized model is in the form of a report shown in Figures 15a, 15b, and 15c. The report contains the various statistics concerning the teletypes, CPU, the selector channels, the bufferpools and the various queues.

It may be pointed out that the CPU service related to interrupts is considered as overhead. Since all the CPU services in the simulated system were represented by interrupts, the generalized model reports that all the CPU utilization was for overhead.

TABLE I

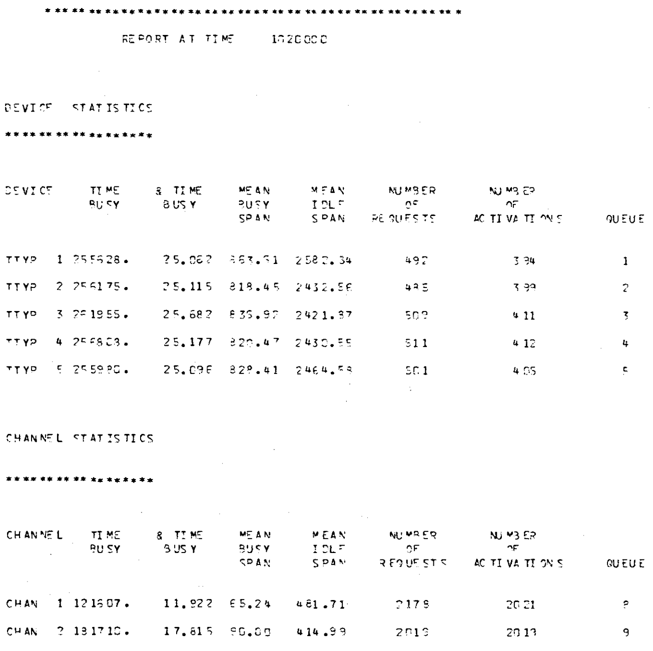
EQUIPMENT		AVERAGE UTILIZATION	
GPSS Model	Generalized Model	GPSS Model	Generalized Model
TERM1	TTYP 1	.242	.2506
TERM2	TTYP 2	.231	.2512
TERM3	TTYP 3	.251	.2568
TERM4	TTYP 4	.266	.2517
TERM5	TTYP 5	.260	.2510
CHANI	CHAN 1	.118	.1192
CPU	PROCESSING UNIT 1	.645	.6385
CHANO	CHAN 2	.177	.1782
CORE	BUFFERPOOL 1	.099	.1010

TABLE II—Message Transit Time Distribution

Interval	Percent of Total	
	GPSS Model	Generalized Model
0-400	10.89	13.026
400-500	16.14	17.385
500-600	20.04	18.722
600-700	11.69	9.807
700-800	9.34	8.370
800-900	7.79	6.984
900-1000	6.04	6.043
1000-1500	12.75	12.927
1500-2000	3.35	4.160
2000-2500	1.30	1.535
2500-3000	.21	.545
3000-3500	.00	.347
3500-4000	.04	.149
over 4000	.00	.000

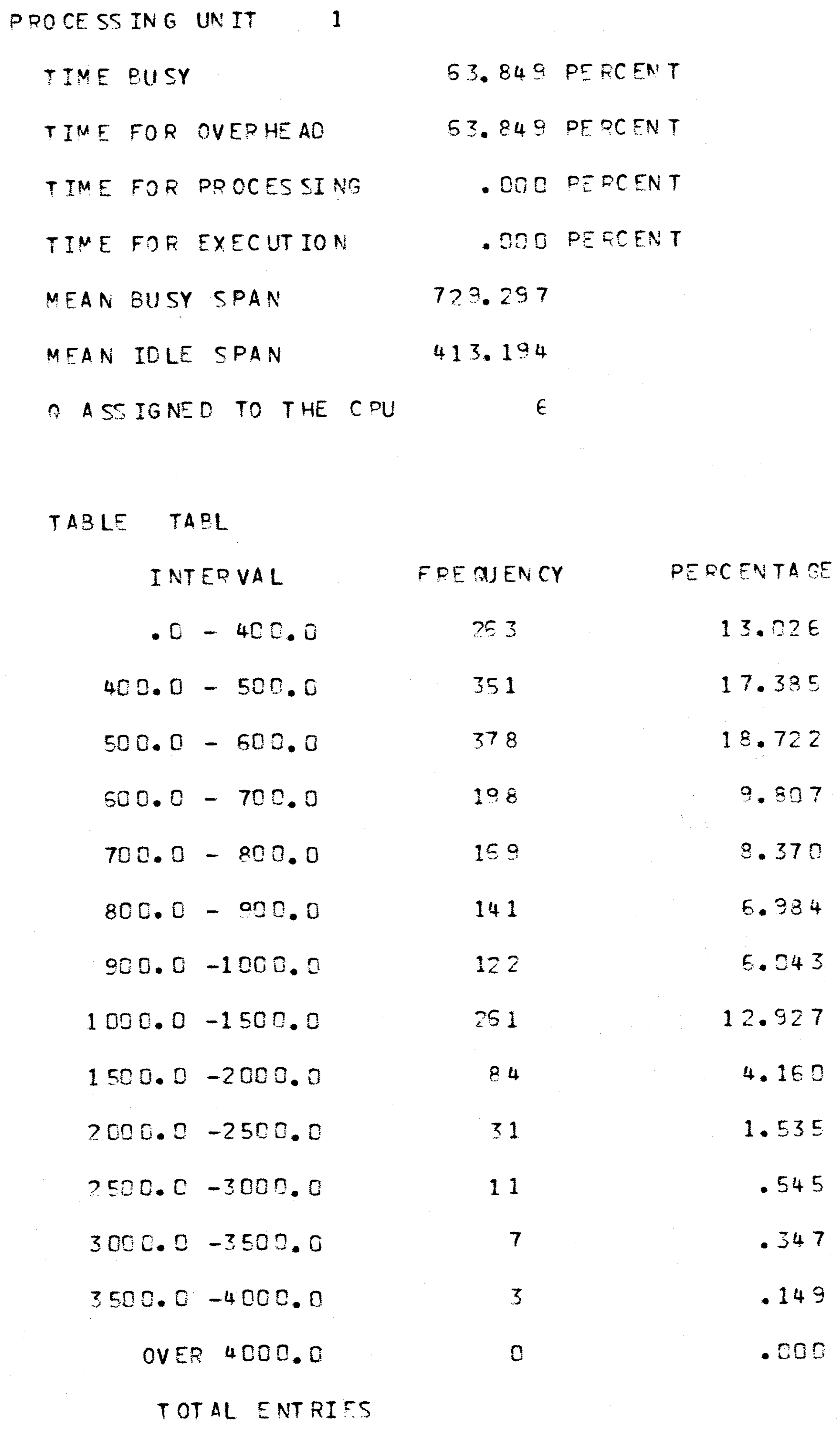
The results obtained by the generalized model are essentially the same as those obtained by the GPSS model. The equipment utilization as reported by the two models is shown in Table I.

The total transit time distribution of the messages as obtained by the two models is shown in Table II. The differences in results obtained by the two models may be attributed to sampling error.



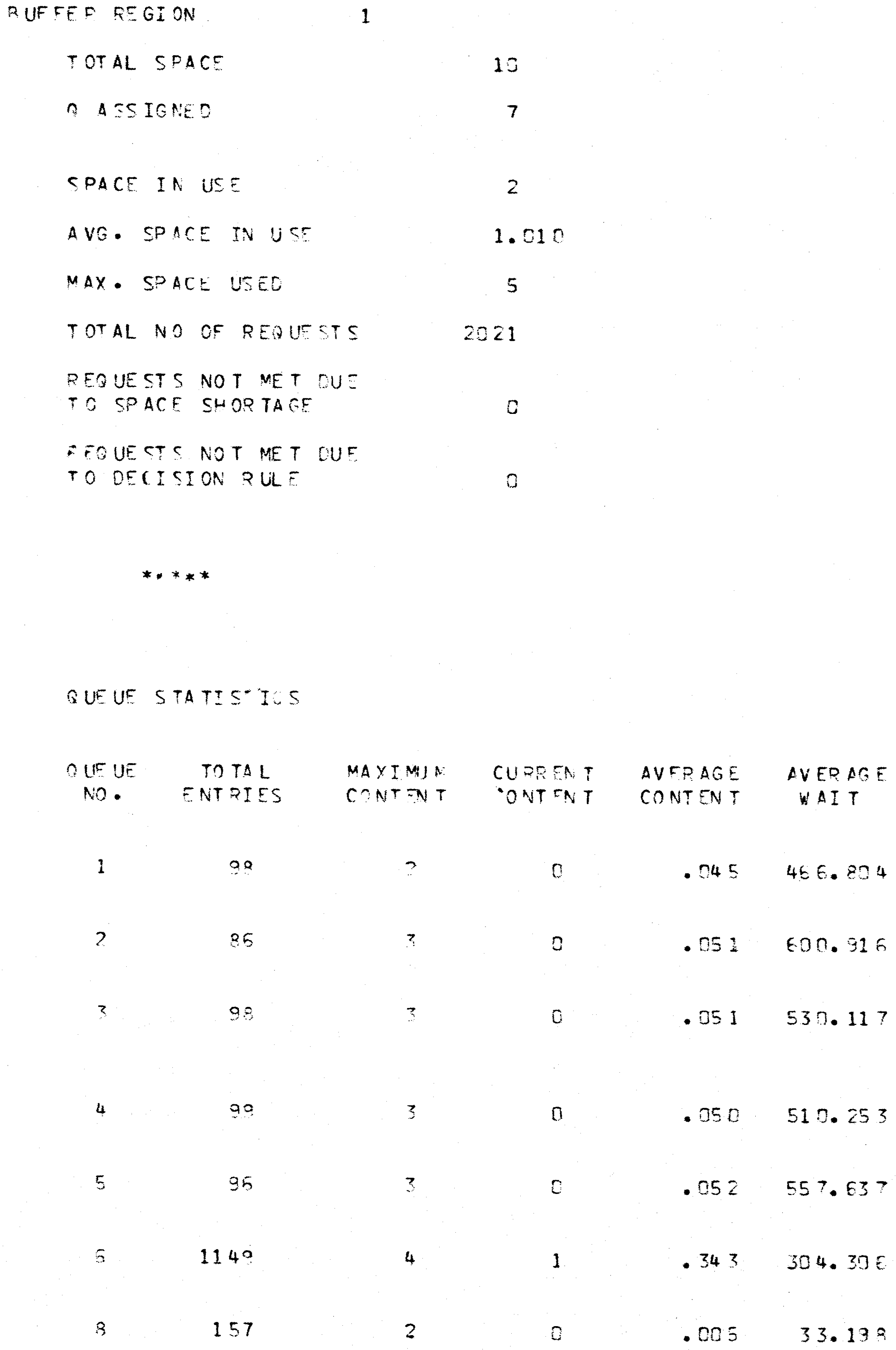
```
***************************************************
           REPORT AT TIME      1020000


DEVICE   STATISTICS
*****************


DEVICE     TIME     % TIME    MEAN     MEAN     NUMBER       NUMBER
           BUSY     BUSY      BUSY     IDLE       OF           OF
                              SPAN     SPAN    REQUESTS    ACTIVATIONS     QUEUE

TTYP  1 255628.    25.062   863.51  2582.34    492          394            1
TTYP  2 256175.    25.115   818.45  2432.56    485          399            2
TTYP  3 261955.    25.682   835.97  2421.37    509          411            3
TTYP  4 256803.    25.177   820.47  2430.55    511          412            4
TTYP  5 255990.    25.096   829.41  2464.58    501          405            5


CHANNEL STATISTICS
*****************


CHANNEL    TIME     % TIME    MEAN     MEAN     NUMBER       NUMBER
           BUSY     BUSY      BUSY     IDLE       OF           OF
                              SPAN     SPAN    REQUESTS    ACTIVATIONS     QUEUE

CHAN  1 121607.    11.922    65.24   481.71    2178         2021           8
CHAN  2 181710.    17.815    90.00   414.99    2019         2013           9
```

Figure 15a—Results of experiment 1



```
PROCESSING UNIT      1

   TIME BUSY                     63.849 PERCENT

   TIME FOR OVERHEAD             63.849 PERCENT

   TIME FOR PROCESSING             .000 PERCENT

   TIME FOR EXECUTION              .000 PERCENT

   MEAN BUSY SPAN               729.297

   MEAN IDLE SPAN               413.194

   Q ASSIGNED TO THE CPU              6


   TABLE   TABL

            INTERVAL             FREQUENCY           PERCENTAGE

            .0 -  400.0             263                13.026

        400.0 -  500.0              351                17.385

        500.0 -  600.0              378                18.722

        600.0 -  700.0              198                 9.807

        700.0 -  800.0              169                 8.370

        800.0 -  900.0              141                 6.984

        900.0 -1000.0               122                 6.043

       1000.0 -1500.0               261                12.927

       1500.0 -2000.0                84                 4.160

       2000.0 -2500.0                31                 1.535

       2500.0 -3000.0                11                  .545

       3000.0 -3500.0                 7                  .347

       3500.0 -4000.0                 3                  .149

            OVER 4000.0               0                  .000

             TOTAL ENTRIES
```

Figure 15b—Results of experiment 1



```
BUFFER REGION             1

     TOTAL SPACE                   10

     Q ASSIGNED                     7


     SPACE IN USE                   2

     AVG. SPACE IN USE              1.010

     MAX. SPACE USED                5

     TOTAL NO OF REQUESTS        2021

     REQUESTS NOT MET DUE
     TO SPACE SHORTAGE              0

     REQUESTS NOT MET DUE
     TO DECISION RULE               0



           *****




     QUEUE STATISTICS


     QUEUE      TOTAL      MAXIMUM     CURRENT     AVERAGE     AVERAGE
      NO.      ENTRIES     CONTENT     CONTENT     CONTENT      WAIT


       1          98          2           0          .045      466.804

       2          86          3           0          .051      600.916

       3          98          3           0          .051      530.117


       4          99          3           0          .050      510.253

       5          96          3           0          .052      557.637

       6        1149          4           1          .343      304.306

       8         157          2           0          .005       33.198
```

Figure 15c—Results of experiment 1

In developing the OSSL version of the GPSS model, the following advantages of the OSSL language over the GPSS language for computer system simulation were noted:

1. There is a more direct correspondence between the various components of a computer system and the available OSSL language segments. This feature practically eliminates the translation of the computer system operation concepts into simulation concepts as contained in GPSS or any other general purpose simulation package.
2. The OSSL language elements reflect the terminology used by the designers and implementors of computer systems. This feature goes a long way in reducing the time required for structuring and debugging of a simulation model. A side benefit is expected to be the ability to involve interested people who do not know the simulation techniques.
3. The OSSL language is self documenting and thus considerably reduces the need for lengthy explanations for communicating the model being simulated.
4. The OSSL simulator is capable in dealing with fractions of unit time, whereas GPSS deals in integral values for time. This feature eliminates the scaling of the time specifications by the user.
5. The OSSL language provides the user the ability to define the various distribution functions in terms of their parameters rather than via a table.
6. The OSSL language provides for free-format input as opposed to the fixed-field format of GPSS.
7. The modularity of a particular simulation model expressed in the OSSL language will allow for several persons to contribute to a simulation study much more easily than the GPSS language.

CONCLUSIONS

The OSSL language, developed for simulating computer systems, seriously attempts to reflect the terminology used by the designers and implementers of computer systems. This semantic structure should allow the use of the language even by those who are not particularly well-versed in the art (and science) of simulation. The free-format syntax of the language should also encourage its use. The OSSL simulator is also inexpensive to use. The simulation run relative to the on-line inquiry system consumed 30 seconds on the UNIVAC 1108 computer system. Other, more complex systems have been simulated in from approximately one to five minutes (CPU time) on the UNIVAC 1108. The ratio of simulated time to real time is not readily generalizable since it depends heavily on the complexity and number of concurrent processes being simulated.

A simulation model structured by the use of the language consists of several interrelatable but modular segments. This modularity makes it easy to effect changes in the model. Moreover, it enables several persons to contribute to the simulation study. In addition to providing simulation capability, the generalized model provides a tool to document computer systems in a relatively unambiguous manner as demonstrated in the examples. This capability, hopefully, will raise the level of communications between personnel dealing with the different aspects of computer systems.

The language may also be used as a pedagogical device in the area of computer operating systems. Modelling of operating systems has the potential of providing to students invaluable insight into the dynamic relationships of various elements of an operating system. For example, the OSSL language is being used in a graduate course in computer operating systems design and in graduate research at the University of Houston. The ability of students without formal training in simulation to grasp, learn, and use the OSSL language in computer systems simulation based on this limited experience has been very gratifying. With the OSSL User's Guide[25] as a reference, students with a strong background in computer operating systems are able to have simple models developed and executing in a matter of days. Relatively complex models have been developed and experiments implemented by a different group of similarly prepared students within a matter of three to four weeks. A more complete analysis of the results of the usage of OSSL in the tracking of computer operating systems topics is being prepared for a future paper.

During the brief but intensive period of usage of OSSL since June of 1971, a list of improvements and modifications is being compiled and some such changes are being implemented as time permits. A partial list of these improvements is as follows:

1. Introduction of dynamic storage allocation for storage relative to model parameters during execution.
2. Provision for more extensive and specific error messages for both compile time and execution time errors.
3. Provision of recursive "Subroutine-like" capa-

bility for procedures written in the OSSL language.

4. Expansion of memory management and processor simulation capabilities to allow for user representation of more "exotic" computer system architectures.

It is recognized that the language may not be able to simulate all computer systems, primarily due to the current weakness of the language in the area of "unconventional" memory management strategies and intraprocessor scheduling algorithms. This weakness suggests that more research needs to be done in this area. To this end, the language has been implemented with an open-ended structure, which will hopefully allow for the inclusion of these and other developments with minimum difficulty.

It is hoped that the language will be used extensively to establish its usefulness as well as its limitations.

REFERENCES

1 L J COHEN
System and software simulator
Defense Documentation Center Number AD 679269 December 1968

2 E G COFFMAN
Stochastic models of multiple and time-shared computer operations
PhD Dissertation Department of Engineering University of California at Los Angeles June 1966

3 C J CONTI D H GIBSON S H PITKOWSKY
Structural aspects of system/360 model 85 I—General organization
IBM Systems Journal Vol 7 Number 1 1968

4 C E DONAGHEY JR
A generalized material handling simulation system
SRCC Report Number 69 University of Pittsburgh Pittsburgh Pennsylvania February 1968

5 *General purpose simulation system/360 introductory user's manual*
IBM Inc Form GH20-0304-4

6 J H KATZ
An experimental model of system/360
Communications of the ACM Vol 10 November 1967

7 J S LIPTAY
Structural aspects of the system/360 Model 85, II-The Cache
IBM Systems Journal Vol 7 Number 1 1968

8 M H MacDOUGALL
Computer system simulation: An introduction
Computing Surveys Vol 2 Number 3 September 1970

9 H M MARKOWTIZ B HAUSNER H W CARR
SIMSCRIPT: A simulation programming language
Prentice Hall Englewood Cliffs New Jersey 1969

10 R L MATTSON J GECSEI D R SLUTZ I L FRAIGER
Evaluation techniques for storage hierarchies
IBM Systems Journal Vol 9 Number 2 1970

11 R M MEADE
On memory system design
1970 Fall Joint Computer Conference Proceedings Vol 37 AFIPS Press Montvale New Jersey

12 R A MERIKALLIO F C HOLLAND
Simulation design of a multiprocessing system
Proceedings of 1968 FJCC Thompson Book Company Washington DC December 1968

13 J H MIZE J G COX
Essentials of simulation
Prentice Hall Inc Englewood Cliffs New Jersey 1968

14 T H NAYLOR J L BALINTFY D S BURDICK K CHU
Computer simulation techniques
John Wiley and Sons Inc 1966

15 N R NIELSEN
The simulation of time-sharing systems
Communications of the ACM Vol 10 No 7 July 1967

16 N R NIELSEN
Computer simulation of computer system performance
Proceedings of 22nd National Conference ACM Thompson Book Company Washington DC August 1967

17 N R NIELSEN
An approach to the simulation of time-sharing systems
Proceedings of the 1967 FJCC Thompson Book Company Washington DC December 1967

18 N R NIELSEN
ECSS: An extendable computer system simulator
Memorandum RM-6132-NASA The Rand Corporation Santa Monica California February 1970

19 S L REHMANN S G GANGEWARE JR
A simulation study of resource management in a time-sharing system
Proceedings of 1968 FJCC Thompson Book Company Washington DC December 1968

20 S ROSEN
Electronic computers: A historical survey
Computing Surveys Vol 1 Number 1 March 1969

21 R F ROSIN
Supervisory and monitor systems
Computing Surveys Vol 1 Number 1 March 1969

22 A L SCHERR
An analysis of time-share systems
PhD Dissertation Department of Electrical Engineering Massachusetts Institute of Technology June 1965

23 T J SCHRIBER
General purpose simulation system/360—Introductory concepts and case studies
University of Michigan Ann Arbor Michigan September 1968

24 P H SEAMAN R C SOUCY
Simulating operating systems
IBM Systems Journal Vol 8 Number 4 1969

25 J B WYATT P B DEWAN
OSSL—Operating systems simulation language—A user's guide
University of Houston Press November 1971

Discrete computer simulation—Technology and applications—The next ten years

by JOHN NORRIS MAGUIRE

Consultant
Reston, Virginia

Digital computer simulation is an effective method of pretesting proposed systems, plans or policies prior to the development of expensive prototypes, field tests or actual implementations. In simulation analysis the computer traces out in detail the implications and consequences of a proposed system or course of action. Compared with other forms of analysis, simulation is more realistic, more easily understood and more conclusive. Because the results are easier to understand, the conclusions receive wider acceptance.

As an aid in making important decisions, the use of computer simulation is growing at an astonishing rate. A decade ago it was used occasionally in manufacturing, military, nuclear, and a few other pioneering applications. In more recent years its use has expanded exponentially, both through increased coverage of old areas and in application to new areas. The growing list of successful uses includes the simulation of proposed information systems and manufacturing facilities; proposed inventory control systems; war games such as air battles, tank battles, and amphibious operations; hydroelectric, transportation, and communication systems; and many more.

Five factors have triggered this growth in the past and will continue to do so in the future. The first is the increasingly widespread recognition of simulation's role in careful planning. It pretests proposals under alternate contingencies prior to implementation. To a certain extent, it provides hindsight in advance.

The second factor contributing to the growing use of simulation is the advancement in techniques for producing simulation models and programs.

The third factor is the increasing availability and use of generalized models which focus on particular problem areas. The most notable successes here have been in the information systems area, but the development and use of generalized models have been spreading rapidly into other areas such as inventory, transportation, logistics, financial and management planning systems.

The fourth factor contributing to the increased use of simulation is the application and acceptance of more scientific techniques to the experimental design aspects of simulation experiments. Coupled with this has been the development of automated analysis techniques for simulation models.

The fifth factor is the increasing availability of data vital to the formulation of models. Naturally, this change has come about mostly in the areas which have been susceptible to a high degree of automation, e.g., computerized information, inventory, financial and manufacturing systems. Successes in these areas have accelerated simulation research and applications in areas such as marketing, social, biomedical, oceanography and environment/ecological systems where the data is more sparse, but growing rapidly.

These statements are generally true for all types of modelling. It is convenient to classify simulation models into two major types: continuous change models and discrete change (or discrete event) models. Some models are clearly best described by one type or the other; for a few problems, either type may be used. Many of the comments to follow on technological developments and trends will be applicable to both kinds of modelling, but the discussion will concentrate on discrete event modelling.

THE SIMULATION PROCESS

A brief review of the simulation process itself is appropriate prior to discussing the recent and likely future technological changes in this process. One view of the simulation process suggests that there are ten steps as illustrated in Figure 1.

1. *Formulate the Problem* This includes a definition of the general objectives of the study; specification of the questions to be answered; and a description of decision criteria and processes.

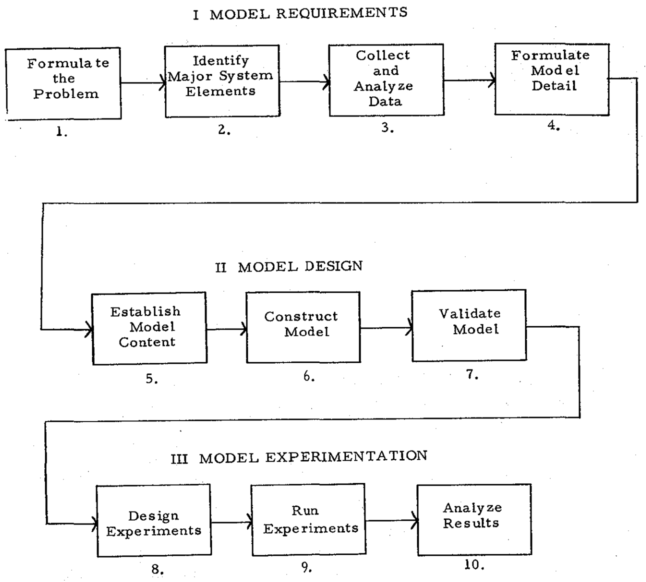


Figure 1—Modelling tasks

2. *Identify Major System Elements* A description of the major system components and features has to be developed; explaining functional dependencies and interrelationships. Also, the "domain" of the system under study should be defined in terms of exogenous and endogenous phenomena.
3. *Collect and Analyze Data* This involves the classification and identification of relevant data sources. Procedures/data collection forms/computer programs/devices may be used in the collection of this data.
4. *Formulate Model Detail* Create the design specifications defining basic elements and their characteristics coupled with a verbal description of model operation.
5. *Establish Model Content* Specify the degree to which phenomena will be enacted. Establish control and experiment parameters along with system performance measures.
6. *Construct Model* Evaluate alternative modelling techniques; flow chart, code and de-bug the model.
7. *Validate Model* Compare simulated data with hypothesized, actual, and/or historic data; and perform sensitivity analysis.
8. *Design Experiments* Evaluate alternative experimental design techniques and compare with SimOptimization* procedure; make decision and set up experiments.
9. *Run Experiments* Implement model initialization and operating procedure and perform experiments.
10. *Analyze Results* Relate the results back to the real phenomena; decide if further experimentation is required; and document results.

Of these ten steps of specified simulation activity, step Number 6, Model Construction, is the one that has seen the most dramatic change in the past decade through the development of high level simulation languages and generalized models. Changes will continue to take place in this area and these will be discussed in a later section of this paper.

Significant changes have also taken place in the following steps:

3. Data Collection
8. Experiment Design
10. Analysis

It is in these three areas where some substantial changes may be expected to occur in the coming decade.

Again, the five factors which seem to have brought about an expotential increase in the use of simulation are:

1. Recognition of simulation's advantages.
2. Advancements in techniques for producing models.
3. Increased availability of generalized models.
4. Increased application of experimental design and automated analysis techniques.
5. Increased availability of relevant model data.

Trends in these five areas can be discussed and extrapolated with a reasonable degree of confidence for the next decade. Other positive factors will also influence the growing use of simulation. These factors include:

- Larger numbers of students leaving the campus trained in simulation techniques.
- Successes with simulation in new application areas.
- Reduction in the cost of using computer simulation, especially for machine time as we enter the "no-cost processor" era in the 1980's.

The major factor on the negative side which might tend to inhibit the growing use of simulation in a particular functional area or organization is either the misapplication of the technique or low quality work performed by an over zealous, ill equipped individual or group. This will happen, as it has in the past, as a field turns from more of an "art" to more of a "science." Examples of this type appear in every new field, be it process control computers or heart transplants, but it is

* SimOptimization is a Trademark and Service Mark of Consolidated Analysis Centers Inc.

not expected to be a significant deterrent to increased use of computer simulation.

1. Recognition of Simulation's Advantages

One indicator that this has occurred is simply the vast sums of money and resources being expended in the area and the trend is clearly up—after possibly a leveling off during the recent recession. One measure of this effort is the published computer simulation papers and conferences devoted solely to the subject. In addition to many simulation papers presented as parts of many conferences such as this Spring Joint Computer Conference during the past five years, new annual simulation conferences include:

- Summer Computer Simulation Conference (1970 Denver, Colorado—1971 Boston, Massachusetts)
- Applications of Simulation (Usually New York)
- Annual Simulation Symposium (Tampa, Florida)

Attendance at these conferences has been growing and the proceedings from the 1971 SCSC ran 1,323 pages. Various organizations and societies have co-sponsored one or more of the above conferences. These societies include:

- American Institute of Aeronautics and Astronautics (AIAA)
- American Institute of Chemical Engineers (AIChE)
- American Institute of Industrial Engineers (AIIE)
- American Meteorological Society (AMS)
- Association for Computing Machinery (ACM)
- Board of Simulation Conferences (BSC)
- Institute of Electrical and Electronics Engineers (IEEE)
- Instrument Society of America (ISA)
- Simulation Councils, Inc. (SCi)
- SHARE—IBM User Group
- The Institute of Management Science (TIMS)

While there is a way to go before operating personnel use, or at least endorse simulation techniques, the previous paragraphs do indicate that general acceptance of the technique has occurred among professionals working in a variety of fields.

2. Advancements in Techniques for Producing Models

These techniques have centered around the development of high level simulation languages. A detailed discussion of these languages is presented in Mr. Ira Kay's paper for this session.

The two most widely used discrete event simulation languages are SIMSCRIPT[1,2,3] and GPSS.[4] A recent development has been the availability of the HOCUS modelling method.[5] With simulation techniques and languages, as with most other tools, there is a trade-off between performance and simplicity of design. The HOCUS approach occupies a particular point on the trade-off curve. SIMSCRIPT and GPSS represent two other distinct trade-off points. Each of the three systems fulfills an important need not fully met by the other two, even though the basic principles underlying all three systems are essentially the same.

The extreme simplicity of HOCUS gives it many important practical advantages over other methods, including quick model formulation, the ability to operate the model by hand, and easy transfer to a computer. On the other hand, some simulation models cannot be forced into the limited HOCUS framework.

At the other extreme, SIMSCRIPT is the most general and powerful of the three techniques. It demands highly skilled simulation analysts or programmers, but its great power and generality are sometimes indispensable. GPSS falls between HOCUS and SIMSCRIPT (Reference No. 6 details an evaluation of major simulation languages—although published in 1966, the study is still reasonably valid). Some systems can be classified as a restricted language or a generalized model depending upon your viewpoint. An example of this is ECSS (see: Kosy, D. W., "Experience with the Extendable Computer System Simulator," RAND No. R-560-NASA/PR, December, 1970).

The main point is that the programming "bottleneck" in the development of simulation models has, for the most part, been solved. Refinements will continue in the simulation language field. Two of the most promising developments are interactive modelling and simulation oriented graphics.

The best known interactive simulation language is OPS-3.[7] This language was designed and implemented at M.I.T. and used successfully to demonstrate the feasibility of interactive modelling. Operating in a time-shared environment under the M.I.T. Compatible Time Sharing System (CTSS), OPS-3 provides a user with on-line interactive communication between himself and a programming system. Using it, one can, in a single sitting, compose a program, test and modify it, expand and embellish it, and prepare it for subsequent production use.

A considerable amount of simulation-oriented graphics research is going on right now. At the Norden Division of United Aircraft, an IBM 2250 is used to modify source language GPSS programs and view their output in graphical form.[8] At the RAND Corporation, the Grail system and the RAND Tablet are used to construct GPSS programs on-line from hand-drawn flowcharts.[9] At M.I.T., the SIMPLE project is using graphics for man-computer interaction during modelling and experimentation. Papers have been written on the use of graphics in simulation modelling and the use of existing graphics packages for analyzing simulation-generated output.[10]

There is little doubt that interactive modelling is carried out better with a CRT device than with a typewriter or line printer. When designing programs, flow-charts and other symbolic notation can be displayed. For analyzing performance data, graphs provide more insight than do lists of numbers. The growing use of line-printer plotting simulators testifies to the utility of graphic output.

Also, we will see during the next decade extensions of a few simulation languages that will include mass storage random access data handling capabilities. There already exists a design and a tested prototype for this in SIMSCRIPT. The model builder will be able to use the simulation language itself to acquire most, if not all, of the data he requires for model formulation and testing.

3. Increased Availability of Generalized Models

During the early 1960's, a few simulators for discrete, dynamic models of manufacturing, inventory and EDP systems appeared and were parameterized and relatively easy to use, but had a narrow range of application. The alternative programming solution for a computer model was frequently machine language or a language such as FORTRAN or ALGOL which typically consumed a large amount of time for implementation. After that period, there emerged a number of simulation languages that attempted to provide the flexibility of machine language with the power of a generalized model. In parallel with these developments, a number of generalized models became available. SCERT,[11] CASE,[12] S.A.M.,[13] LOMUSS,[14] and ECSS[15] are examples of generalized models of EDP systems. These models possess the advantage of "ease of use" if their structure happens to fit your EDP modelling problem. Examples of their general use are numerous,[16,17] and growing rapidly. Successful simulation applications of generalized models have helped to determine EDP design answers such as machine core size, secondary storage assignments, peripheral terminal configuration, operating system procedures and allocation rules. Similar successful use of generalized models in other problem areas insure their increased use, because of the rapid implementation and low cost usually experienced with each application of a generalized model.

4. Increased Application of Experimental Design and Automated Analysis Techniques

Much has been written about the use of computer simulation experiments for the study and design of systems.[18] For the most part, the literature has emphasized the "model building" aspects rather than the experimental design aspects of simulation experiments. It may be that by the time the typical model-builder has gone through all the steps of implementing a model, he has run out of time and frantically makes a series of computer runs changing parameters here and there on a "gut-feeling" basis. He then culls through stacks of computer output trying to perform analysis. Other factors presently associated with this situation of not using more sophisticated techniques may be *cost* (these techniques tend to require more labor and computer time) and training prerequisites (A strong mathematical and/or statistical background is required.) Nevertheless, increasing use of more analytical techniques for simulation of systems is a near certainty.

In a well-designed experiment, consideration must be given to methods of analyzing the data once it is obtained. Most of the classical experimental design techniques described in the literature (e.g., 19, 20) are used in the expectation that the data will be analyzed by one or both of the following two methods: analysis of variance and regression analysis. Recently, however, several other techniques have been proposed for analyzing data generated by computer simulation models including multiple ranking procedures, sequential sampling plans and time series analysis. Also, a technique called "SimOptimization" shows promise for automating and minimizing the number of simulation runs required.

Analysis of variance

The analysis of variance is a collection of techniques for data analysis which are appropriate when qualitative factors are present, although quantitative factors are not excluded. An example of a qualitative factor would be the testing of different decision rules in a simulation model. Regression analysis is a collection of techniques for data analysis which utilizes the numerical properties of the levels of quantitative factors. The parameters chosen for describing probability distributions are quantitative factors, but the type of probability distribution (e.g., exponential or normal) would be a qualitative factor. From a mathematical point of view the distinction between regression and analysis of variance is somewhat artificial. For example, an analysis of variance can be performed as a regression analysis using dummy variables which can assume only the values of zero or one. A treatment of the relationship between regression and the analysis of variance can be found in the book by Graybill.[21]

Multiple ranking procedures

Conway[22] and others have pointed out that analysis of variance techniques frequently do not supply the kind of information that the experimenter seeks. Referring specifically to the design of computer simulation experiments, Conway has stated that:

> *the analysis of variance seems a completely inappropriate approach to these problems. It is centered upon the test of the hypothesis that all of the alternatives are equivalent. Yet the alternatives are actually different and it is reasonable to expect some differences in performances, however slight. Thus the failure to reject the null hypothesis only indicates that the test was not sufficiently powerful to detect the difference—e.g., a longer run would have been employed. Moreover, even when the investigator rejects the hypothesis, it is highly likely that he is more interested in identifying the best alternative than in simply concluding that the alternatives are not equivalent. Recently proposed ranking . . . seem more appropriate to the problem than the conventional analysis of variance techniques, but the investigator is still going to have difficulty satisfying the assumptions (normality, common variance, independence) that the statistician will require.*[22(p. 53)]

Bechhofer's procedure for ranking means of normal populations with known variances[23,24] and Bechhofer and Sobel's procedure for ranking variances of normal populations[25] are examples of multiple ranking procedures which seem appropriate to the design of computer simulation experiments.

Sequential sampling plans

Experimental designs which use analysis of variance as a technique of data analysis are based upon the assumption of fixed sample size. However, with certain kinds of experiments conducted on an "accumulation-of-information" basis, sequential sampling methods can be utilized which may lead to significant reductions in sample size. Since we are using computers on an accumulation-of-information basis and not planting plots of beans that will mature next year, we may want to take advantage of the savings (in terms of computer time) sequential experiments offer. Sequential sampling methods were designed for testing hypotheses or estimating parameters "when the sample size is not fixed in advance but is determined during the course of the experiment by criteria which depend on the observations as they occur."[26(p.365)]

In testing a hypothesis with a computer model, the sequential method yields a rule for making one of the following three decisions at each stage of the simulation experiment:[27(p.271)] (1) Accept the null hypothesis; (2) Reject the null hypothesis; (3) Continue the simulation experiment by generating additional data.

Although Wald's *Sequential Analysis*[28] is probably the best known reference on sequential sampling procedures, the optimization procedures developed by Dvoretzky, Kiefer and Wolfowitz[29(pp.51–55)] appear to offer promise in analyzing data generated by computer models. Chernoff's[30] article entitled, "Sequential Design of Experiments" considers a number of important aspects of sequential designs also.

Spectral analysis

Spectral Analysis[31] is a statistical technique widely used in the physical sciences to analyze time-dependent physical processes. There are at least two reasons why one may want to consider spectral analysis as a technique for analyzing data generated by computer simulation models. First, data generated by computer simulation experiments are usually highly autocorrelated. Yet classical statistical theory of the design of experiments (analysis of variance) is based on the assumption that component observations or experimental outcomes are independently distributed. In simulation experiments involving autocorrelated data, classical statistical theory must be replaced by a sampling theory such as spectral analysis in which the probabilities of component outcomes in a series depend on other outcomes at other time points in the series. Second, as one becomes more sophisticated in the analysis of computer simulation data, he may become interested in analyzing more than expected values and variances. "When one studies a stochastic process, he is interested in the average level of activity, deviations from this level, and how long these deviations last, once they occur. Spectral analysis provides us with this kind of information. Spectral analysis studies the salient time properties of a process and presents them in an easily interpretable fashion for descriptive and comparative purposes."[32]

Fishman and Kiviat[32] have written a survey article on the use of spectral analysis in analyzing data generated by computer simulation models. Tukey[33] has written an article in which spectral analysis and the analysis of variance are compared in detail.

SimOptimization

SimOptimization[34,35,36] is a set of optimum-seeking techniques embedded in an automatic cycle of simulation-analysis-simulation. These procedures quickly locate improved policies and computer system configurations in simulation models. Their use greatly increases the value of simulation analysis.

While simulation is more realistic and easier to understand than mathematical optimization, it has the disadvantage of being somewhat clumsy to use. A typical simulation model has an enormous parameter space in which the desired optimum solution is concealed. Finding this solution can be difficult.

Fortunately, a good analyst, using his judgment and intuition can find improved solutions by studying the results of a limited number of simulation runs. Even so, the process of making a few runs, studying them, deciding what runs to make next, then repeating this procedure, is a time consuming and costly way to grope for a good solution. By using SimOptimization, the preferred solution for most models can likely be determined in a single computer run consisting of from two to four simulation and analysis phases.

In most simulation models, optimality cannot, of course, be proven rigorously. However, SimOptimization solutions are at or very close to the "apparent" optimum. In every case, a "good" solution to a realistic model is preferable to an "optimal" solution to an unrealistic model.

While SimOptimization is still new, all of its applications have been highly successful. Judging from this empirical experience, SimOptimization seems to offer the best of two worlds, namely the realism of digital simulation combined with the convenience of automatic analytical solutions.

5. Increased Availability of Relevant Model Data

The technological changes taking place today in data collection in information processing, or "EDP Systems" is taking place in parallel with other areas such as finance and manufacturing and portends what is going to happen in the future in other areas susceptible to simulation. A detailed discussion follows covering the trends in EDP System data collection. Within the next decade we will see similar changes in other areas where data is now laboriously collected only on an *ad hoc* basis for a simulation study.

A key step in the process of constructing an EDP system model is the collection of data characterizing the work load. Of course, if we are hypothesizing a new system in a new environment, it doesn't pose much of a problem. But, typically, the model builder is studying possible hardware and software changes to an existing system. Therefore, he must represent as accurately as possible the complete system. In the past, this meant collecting some data on the work load and the computer system. Job data was (and still is) frequently obtained from an accounting file generated by the operating system. The characteristics of the hardware are obtained from the technical manuals.* This is the procedure used in applications, of which the study by Hutchinson and Maguire[17] is an example.

But computer system complexity has increased faster than our ability to characterize these systems. Today, the complex interactions between the work load, hardware and software of computer systems make the batch-processing systems of the second generation appear very simple, indeed. The difficulty today of even intuitively being able to predict the performances of some of these systems creates a situation where simulation for EDP systems is in more demand than ever. The recent development of EDP System Hardware and Software monitors offers a solution to this problem. As will be shown, use of these monitors can help solve operating problems and increase efficiency in the short run, but an even greater value may be in their use for generating data for EDP system models for pretesting system changes.

Hardware monitors

A hardware monitor consists of sensors, control logic, accumulators, and a recording unit. The sensors are attached on the back panels of the computer system components—CPU, channels, disks, etc. The signals from registers, indicators and activity lines picked up by the sensors can be logically combined or entire registers can be compared at the control panel and then be routed to the accumulators for combining or timing events, e.g., CPU active, any channel busy, disk seek counts and times, etc. Typically the contents of the accumulators are written periodically to a magnetic tape unit. The magnetic tape is batch-processed by a data analysis program to produce a series of analysis, summary and graphic reports. Figure 2 shows a hardware monitoring system.

* Some generalized models such as SCERT and CASE maintain this data on tape for convenience in processing.

Table I lists some hardware monitors presently on the market.

Figure 3 illustrates some typical output from a data analysis program run against a monitor tape.[37]

Two examples of how hardware monitors are used to improve system performance are:

1. Channel Imbalance

If the channel utilization is high, but the channel load is not balanced, the device activity needs to be measured to determine device rearranging. A new system performance profile should be measured to verify the results.

Low channel utilization would indicate that all the work can be placed on one channel with little effect on the system throughput.

2. Multi-programming Effectiveness

A large CPU ACTIVE ONLY time coupled with a low CPU-CHANNEL OVERLAP indicates that the benefits of multi-programming are not being obtained. Possible reasons are poor job scheduling (a balance is needed between CPU and I/O bound jobs); poor data set allocation (data sets should be on different channels so they do not have to be retrieved sequentially), or inefficiently written programs. Several of the production jobs which use a large part of the system resources should be selected and run stand alone in the system to create an individual job profile. These profiles show data set usage to make data set placement changes and processing phases which are CPU or I/O bound to enable efficient job scheduling. This will increase CPU utilization and system throughput.

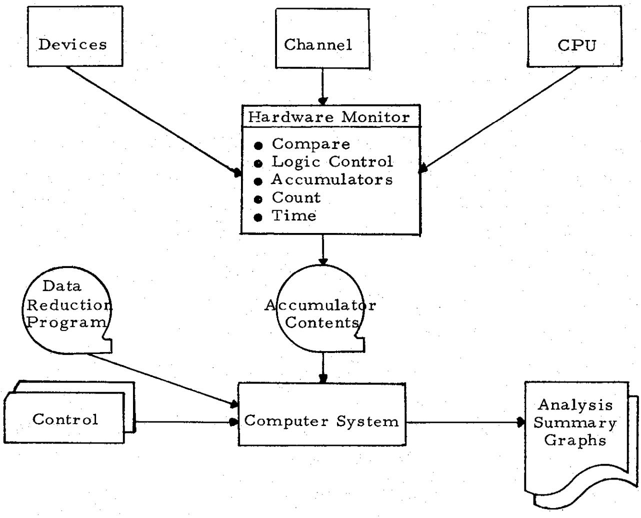


Figure 2—Hardware monitor system

TABLE I—Computer Hardware Monitors—A Sampling*

Supplier	Hardware Monitor	Price
Allied Computer Technology, Inc. 1610 26th Street Santa Monica, California 90404	CPM II	$35,000 to $80,000
Telephone: (213) 828–7471	CPM III	$14,700 to $24,000
Computer Learning & Systems Corp. 5530 Wisconsin Avenue Suite 1600 Chevy Chase, Maryland 20015 Telephone: (301) 652–9220	X-RAY	$2,350/month or $50,0000 purchase
Computer Synectics, Inc. 328 Martin Avenue Santa Clara, California 95050 Telephone: (408) 247–0200	SUM	Starts at $1,750/ month for one year lease or $34,000 purchase
COMRESS 2 Rockville Court Rockville, Maryland 20850 Telephone: (301) 948–8000	Dynaprobe	$4,000/month—lease $34,000—typical purchase price
IBM Corporation 112 E. Post Road White Plains, New York 10601 Telephone: (914) 696–1900	SMIS	$1,800 for basic service contract
Computer & Programming Analysis Inc. One Wynnewood Road Wynnewood, Pa. 19096	CPA 7700	$5,000 to $40,000

* For Suppliers not included or if listed Supplier's information is not up-to-date, please write to author.

While the data derived from the hardware monitor is useful for immediately improving system performance, it can also be very valuable to the model builder contemplating major changes or new systems. This data serves as:

- Input to development of the model.
- A validation "check point" for model testing.

The hardware monitor does not degrade or interfere with the system it is measuring and requires no system overhead. A software monitor, on the other hand, is a

```
Measured 10/6/70         System Model 40 and 50 Analysis CPU Active Model 50
            0  10  20   30    40   50   60   70    80     90   100
  Secs.     x   x   x    x     x    x    x    x     x      x    x     Time
          *******************************************************
  90.0    1111  .   .    .     .    .    .    .     .      .    .     8.31.30.0
 180.0    111111111111   .     .    .    .    .     .      .    .     8.33. 0.0
 270.0    1111111111111111111111    .    .    .     .      .    .     8.34.30.0
 360.0    11111111111111.      .    .    .    .     .      .    .     8.36. 0.0
 450.0    1111111111111 .      .    .    .    .     .      .    .     8.37.30.0
 540.0    11111111111111111    .    .    .    .     .      .    .     8.39. 0.0
 630.0    11111.    .    .     .    .    .    .     .      .    .     8.40.30.0
 720.0    11111111 .     .     .    .    .    .     .      .    .     8.42. 0.0
 810.0    11111111111111111111111   .    .    .     .      .    .     8.43.30.0
 900.0    1111111111111111     .    .    .    .     .      .    .     8.45. 0.0
 990.0    111111111111   .     .    .    .    .     .      .    .     8.46.30.0
1080.0    1111111111111111     .    .    .    .     .      .    .     8.48. 0.0
1170.0    1111111   .    .     .    .    .    .     .      .    .     8.49.30.0
1260.0    11111111111    .     .    .    .    .     .      .    .     8.51. 0.0
1350.0    111111111111111111111111111111.     .     .      .    .     8.52.30.0
1440.0    1111111111111111111111111111111111111     .      .    .     8.54. 0.0
1530.0    1111111111111111111111111111111111  .     .      .    .     8.55.30.0
1620.0    1111111111111 .      .    .    .    .     .      .    .     8.57. 0.0
1710.0    11111111111    .     .    .    .    .     .      .    .     8.58.30.0
1800.0    11111111111    .     .    .    .    .     .      .    .     9. 0. 0.0
          *******************************************************
  Secs.     x   x   x    x     x    x    x    x     x      x    x
            0  10  20   30    40   50   60   70    80     90   100
```

Figure 3—Histogram of CPU active

program itself, uses system resources, and adds to system overhead. Both have their merits. A software monitor costs less than a hardware monitor and does a better job of getting a "fix" on the system workload.

Software monitors

The important variables in a computer system can be classed as either quantitative variables or qualitative variables. Quantitative variables are concerned with the magnitude of some quantity such as cpu active time. Qualitative variables identify which, of a perhaps very large number of different system elements, are being considered. For example, when measuring the work performed by a given program (i.e., the percent of available cpu cycles being utilized), it is necessary to identify each module and overlay segment individually and to show the cpu activity requirements of each. The module and segment names are the qualitative or descriptive variables, the cpu activity associated with each is the quantitative variable.

Qualitative and quantitative variables must be combined to present a usable set of measurements. One of the advantages of software monitors is their ability to obtain both from the system itself. This eliminates the necessity for using manual techniques to obtain or correlate information on what elements were active during the data collection process. When the descriptive data required is not available through console logs or other means, then either a pure software approach must be used or special software supplied to obtain the needed descriptors in a manner which permits proper correlation.

How a software monitor extracts variables from memory is conditioned on the timing and form in which such variables are retrieved in memory, which in turn tends to be a function of the structure of the operating system, compilers, etc. Thus, software monitors are at least operating systems dependent and can be compiler dependent or even problem program dependent.

Ideally, the act of taking a measurement should not alter the system being measured nor require the system to be altered beforehand. This is not perfectly achievable in software monitors. A good software monitor extracts data from the system without *significantly* altering the run characteristics of the system. This involves three parameters: cpu cycle requirements, I/O activity, and core usage. The first two quantitative variables can be expressed as a percentage of some total time, rather than an absolute count. Thus classical sampling techniques can be used instead of activating special codes or going to the techniques of trace routines. Furthermore, if the frequency of sampling is less than the minimum time in which some event can occur, then other measures of interest can also be supplied. Thus, even though the sampling techniques are used, software monitors can obtain the cylinder addresses of successive head movements on discs by using a sample interval less than the minimum positioning time.

The third design parameter—core usage—is resolved typically by the simple expedient of separating the data extractor mechanism from the data analysis function. As an example, Figure 4 illustrates this structure for the Boole and Babbage program evaluator (PPE).[38]

Application examples are given in References 39 and 42 using PPE[40] and another software monitor called CUE.[41]

This has several other important benefits, such as the ability to combine and/or reanalyze raw data in various ways. While some software measurement tools have been built with the extraction and the data reduction mechanisms combined, experience with them has tended to further strengthen the case for separation of these two functions. Table II lists some software monitors presently available.

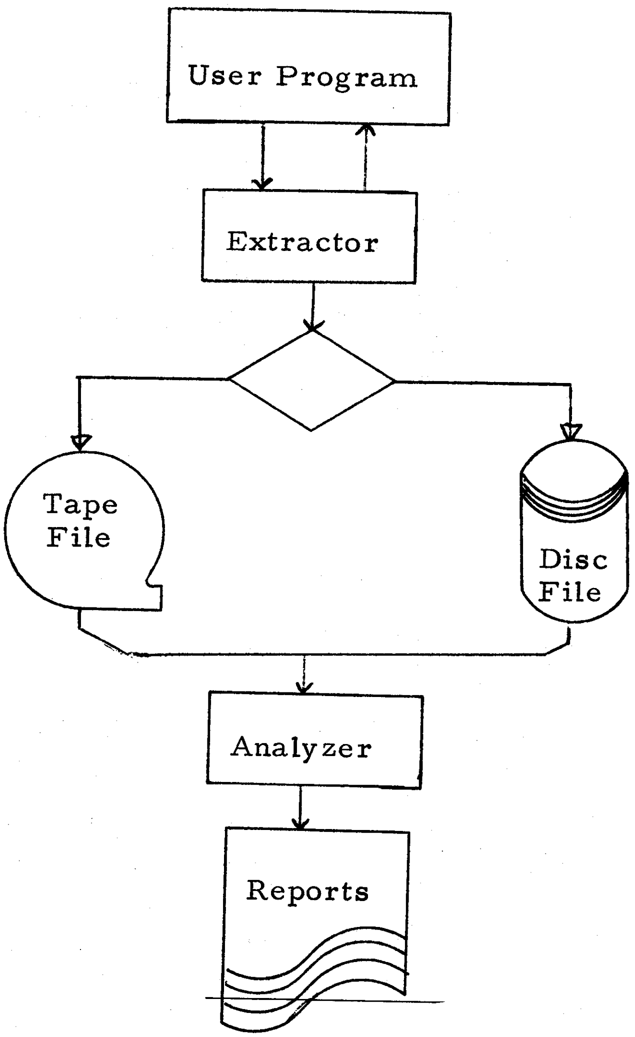


Figure 4—PPE—General procedure

TABLE II—Computer Software Monitors—A Sampling*

Supplier	Software Monitor	Price
Allied Computer Technology, Inc. 1610 26th Street Santa Monica, California 90404 Telephone: (213) 828–7471	CPS II	$6,500
Boole and Babbage, Inc. 1121 San Antonio Road Palo Alto, California 94303 Telephone: (408) 255–1200	PPE or CUE	$8,800 or $15,400 for both
Boothe Resources International 3435 Wilshire Boulevard Los Angeles, California 90005 Telephone: (213) 380–5700	CIMS I or II	$3,500 to $5,800
Computing Efficiency, Inc. 35 Orville Drive Islip, New York 11716 Telephone: (516) 567–3600	Compumeter	$3,500 for 360/DOS $15,000 for 360/OS
Lambda Corporation 1501 Wilson Boulevard Arlington, Virginia 22209	LEAP	$8,500
Webster Computer Corporation 1 Padanaran Road Danbury, Connecticut 06810 Telephone: (203) 744–6500	MURS	Starts at $2,500

* For Suppliers not included, or if listed supplier's information is not up-to-date, please write to author.

The techniques of sampling as used in software monitors are based on relatively straightforward mathematical concepts. Assuming randomness (viz. absence of synchronization) of observation with respect to the events being sampled, the accuracy of the data obtained is a function of the number of samples taken, *not* the frequency of sampling. For example, monitoring a one-hour program with a sample frequency of one-half second, for a total of 7,200 samples, results in data accurate to within 1.5 percent with a confidence level of 99 percent. The cpu cycle needs of an extraction mechanism would be much less than 1 percent in this instance. On the other hand, the "overhead" can run as high as 20 percent with a sample frequency as small as 16 milliseconds.

```
                  Code Execution Frequency for Each Interval

Starting      Ending        Interval    Cumulative  Histogram—% of Time (Each * = 0.5%)
Location      Location      Percent     Percent      .0       4.0       8.0       12.0

000000        00061F         0.00         0.00       _
000620        00068F         7.39         7.39       _**************
000690        0006FF         0.0          7.39       _
000700        00076F         1.56         8.95       _***
000770        0007DF        15.59        24.54       _*******************************
0007E0        00084F        13.94        38.48       _***************************
000850        0008BF        12.58        51.06       _*************************
000800        00092F         2.39        53.45       _****
000930        00099F         3.41        53.86       _
0009A0        000B5F         0.0         53.86       _
000B60        000BCF         1.28        55.14       _**
000BD0        000C3F         1.84        56.98       _***
000C40        000CAF         3.45        60.43       _******
000CB0        000D1F          .30        60.73       _
```

Figure 5—Partial sample of code activity histogram

An example of software monitor output is illustrated in Figure 5.[38]

These monitors are most useful when used on a job mix containing programs which are either run frequently or which take a long time when run. The most effective technique is to maintain a list of the top ten "cpu cycle using" programs, and to continuously work on these to reduce their cpu cycle time requirements. As improvements are made on one, other programs usually then move up the list for rework. This activity coupled with configuration improvements can result in very significant improvements to system throughput rate.

FUTURE TRENDS

The practical application of hardware and software monitors for improving EDP system efficiency has been amply demonstrated in the past year or so. These monitors provide a wide spectrum of detailed information on EDP systems, their workload and the interactions between hardware, software (operating system), and user programs. The potential for using this information for the construction, validation and use of EDP system models has not been realized. There appears to be a need for EDP models for planning purposes or for pretesting system changes suggested by intuition or monitor use. The monitors by themselves have limitations for planning future EDP systems. There are a few indications that the "monitor people" will add some simulation capability to their monitors. For example:

> *Software adds important new capabilities to the hardware monitor, significantly increasing the usefulness, power and flexibility of such systems. The first software specifically designed to simulate hardware changes based on actual measurements made by hardware monitors has just been announced by Computer Synectics, and deserves mention. In addition to the 16 hardware accumulators provided by the hardware monitor, this software provides 19 pseudo-accumulators and time of day. A powerful set of operators is provided to manipulate these pseudo-accumulators. Virtual simulation of a wide range of possible hardware and software alternatives is thus made feasible, which can give rise to marked improvements in overall system performance, but based on recorded data about the user's actual job mix and working environment rather than on theoretical factors alone, and before equipment is physically changed or installed. And these advances are achieved without the inclusion of a minicomputer in the hardware monitor.*[43]

Recent research work by Noetzel[44] conceptualizes a design where a "meta-system" automates most of the simulation functions for a system designer. This concept is illustrated in Figure 6.

The concept for such a system has been fairly well

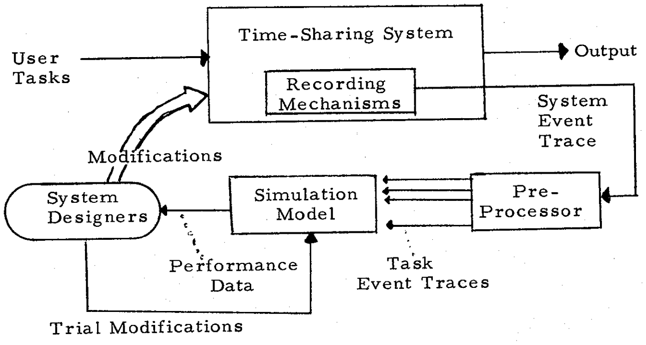


Figure 6—META system design

detailed; the operational demonstration of such a system has yet to be shown; and the economic feasibility of such a system remains dubious.

It appears that the next major step in EDP system design methodology will be widespread use of EDP hardware and software monitors (in addition to their normal operational use) as valuable sources of data for EDP model construction and use. These models will typically be implemented through use of generalized EDP models such as SCERT or CASE, or through use of high level software such as SIMSCRIPT, GPSS, or HOCUS for customized models.

From a simulation viewpoint, these technological changes taking place in the EDP system design area are examples of what is being experienced in other "hard" data areas such as finance, logistics and manufacturing. One can expect similar changes in the "soft" data areas such as marketing, social, and oceanography within the next decade.

In the coming decade, management personnel will be increasingly involved in the development of simulation models to *assist* them in making management decisions at all levels. In the past, the false promise of the computer automating management decision making disillusioned many management personnel who have again relegated the computer to routine data processing.

A pre-determined simple mathematical statement to be minimized or maximized by a mathematical procedure inside a computer has, so far, turned out to be a very poor substitute for the shifting sensitivity of a good business executive to the implications of a decision for many groups. There seems to be no alternative to discussions with a variety of people in order both to consider the trade-offs between various goals of the organization and to get from the various parties the commitment which is essential to the successful implementation of the decision.

But, as standard optimizing procedures, such as linear programming, have been losing favor with management, simulation concepts have been attracting more and more interest—and will continue to do so. The judgment of the manager plays a crucial role in the formulation of most management-oriented models. Many model interrelationships and variables depend upon the intuition, experience and judgment of management. With the growing participation of management in model development and use, further acceptance of the technique and implementation of models at higher levels in organizations is likely to follow.

This trend will be accelerated by an increasing number of models which will be implemented via a terminal into a time-sharing computer system. Often, this terminal will consist of a CRT device with associated software for the display of model output data in graphical forms. These technical innovations will facilitate the direct participation of management in model execution and modification. The next ten years will be an exciting and productive era for simulation techniques.

REFERENCES

1 H MARKOWITZ B HAUSNER H KARR
SIMSCRIPT: A simulation programming language
Prentice Hall Englewood Cliffs NJ 1963

2 P J KIVIAT R VILLANUEVA H M MARKOWITZ
The SIMSCRIPT II programming language
The Rand Corporation R-460-PR October 1968

3 H W KARR H KLEINE H M MARKOWITZ
SIMSCRIPT I.5.
CACI. CACI. 65-INT-1 Santa Monica California June 1965

4 *Users manual, general purpose systems simulator*
IBM Corporation/360 OS Program No 5734-XS1 251 pp 1969

5 P R HILLS T G POOLE
A method of simplifying the production of computer simulation models (HOCUS)
A paper presented to the Institute of Management Sciences Tenth American Meeting Atlanta Georgia October 1-3 1969

6 D TEICHROEW J F LUBIN
Computer Simulation—Discussion of the technique and comparison of languages
Communications of the ACM Vol 9 No 10 October 1966

7 M GREENBERGER M JONES
On line computation and simulation
The MIT Press 1965

8 J REITMAN
GPSS/360 Norden, The Unbounded and Displayed GPSS
Proceedings of SHARE XXX Houston Texas February 29 1968

9 J P HAVERTY
Grial/GPSS, graphic on-line modeling
The Rand Corporation P 3838 January 1968

10 M R LACKNER
Graphic forms for modeling and simulation
Presented at the 1967 IFIP Working Conference on Simulation Languages Oslo Norway

11 R G CANNING
Data processing planning via simulation
(SCERT) EDP Analyzer Vol 6 No 4 pp 1-13 April 1968

12 *CASE—Computer-Aided System Evaluation*
Technical Manual Computer Learning and Systems Corporation/Software Products Corporation 1968

13 L J COHEN
System Analysis Machine-Introductory Guide
P-101-S ADRA upril 1970

14 R W GROTE *et al*
The Lockheed multipurpose simulation system (LOMUSS II)
Lockheed Document M-07-66-1 July 1966

15 N R NIELSEN
ECSS: An extendable computer system simulator
RAND Memorandum No RM-6132-NASA February 1970

16 J W SUTHERLAND
The configurator, today and tommorrow
Computer Decisions pp 38-43 February 1971

17 G K HUTCHINSON J N MAGUIRE
Computer systems design and analysis through simulation
AFIPS Proceedings—Fall Joint Computer Conference Vol 27 pp 161-167 1965

18 D S BURDICK T H NAYLOR
Design of computer simulation experiments for industrial systems
Communications of the ACM Vol 9 No 5 May 1966

19 W G COCHRAN G M COX
Experimental designs
John Wiley New York 1957

20 D R COX
Planning of experiments
John Wiley New York 1958

21 F A GRAYBILL
An introduction to linear statistical models
Vol 1 McGraw Hill New York 1961

22 R W CONWAY
Some tactical problems in digital simulation
Management Science Vol 10 pp 47-61 October 1963

23 R E BECHHOFER
A three decision problem concerning the mean of a normal population
American Statistical Association Meeting New York New York December 30 1955

24 ______
A single sample multiple procedure for ranking means of normal populations with known variances
Annals of Mathematical Statistics Vol 25 pp 16-39 1954

25 ______
A single sample multiple decision procedure for ranking variances of normal populations
Annals of Mathematical Statistics Vol 25 pp 273-289 1954

26 A M MOOD
Introduction to the theory of statistics
McGraw-Hill New York 1950

27 P G HOEL
Introduction to mathematical statistics
John Wiley New York 1954

28 A WALD
Sequential analysis
John Wiley New York 1947

29 A DVORETZKY I KIEFER J WOLFOWITZ
Sequential decision problems for processes with continuous time parameter problems of estimation
Annals of Mathematical Statistics Vol 34 pp 403-415 1963

30 H CHERNOFF
Sequential design of experiments
Annals of Mathematical Statistics Vol 30 pp 770-775 September 1959

31 R A FISHER
The design of experiments
Oliver and Boyd London 1951

32 G S FISHMAN P J KIVIAT
Spectral analysis of time series generated by simulation Models
The Rand Corporation RM-4393-PR February 1965

33 J W TUKEY
Discussion emphasizing the connection between analysis of variance and spectral analysis
Technometrics Vol 3 pp 191-220 May 1961

34 H W KARR et al
SimOptimization research—Phase I
CACI 65-P 2.0-1 November 1965

35 E L LUTHER H M MARKOWITZ
SimOptimization research—Phase II
CACI 69-P2.0-1 July 1966

36 E L LUTHER N H WRIGHT
SimOptimization research—Phase III
CACI 69-P2.0-1 August 1969

37 J S COCKRUM E D CROCKETT
Interpreting the results of a hardware systems monitor
AFIPS Proceedings—Spring Joint Computer Conference Vol 38 pp 23-38 1971

38 K W KOLENCE
A software view of measurement tools
Datamation Vol 17 No 1 pp 32-38 January 1971

39 S C DARDEN S B HELLER
Streamline your software development
Computer Decisions pp 29-33 October 1970

40 *Problem program evaluator (PPE) user's guide*
Boole and Babbage Inc Cupertino California March 1970

41 *Configuration utilization (CUE) user's guide*
Boole and Babbage Inc Cupertino California March 1970

42 B A LICHTIG
Nevada doesn't gamble on their computer, they measure it!
Unpublished Report 30 pp May 1971

43 C D WARNER
Monitoring: A key to cost efficiency
Datamation Vol 17 No 1 pp 40-49 January 1971

44 A S NOETZEL
The design of a meta-system
AFIPS Proceedings—Spring Joint Computer Conference Vol 38 pp 415-424 1971

The emergence of the computer utility

by R. S. MANNA

Robert Manna Associates
Dallas, Texas

and

H. WALDBURGER

University Computing Company
Dallas, Texas

and

D. R. WHITSON

Datotek, Inc.
Dallas, Texas

HISTORY

By 1970 some users had regained control of the computer. This occurred because an operational mode had come into existence that permitted the man and the computer to simultaneously attain the productive heights that each had previously, but separately, achieved. The installation of this mode was prompted by the results of analyzing the computing environment of the late 1950's and early 1960's.

Scientific computer installations circa 1960 operated in an environment in which computer runs, or "shots", were pre-scheduled, making the total system responsive to the individual who had "hands on" control of the machine. He was permitted to stop processor execution at some pre-set memory or instruction cycle, insert changes via the console, manually position tapes, take and immediately print snapshot dumps, relocate I/O channels, and isolate hardware failures. The total resources of the installation were provided to achieve this mode of man and machine interaction that resulted in high user productivity.

Business oriented installations had a less personal environment because of the production, rather than developmental nature of their work. However, production runs did require manual intervention because the processing cycle generally consisted of a progressing chain of separate events. Consequently, in both situations, operators were designated to service the printer, punch, reader, and other peripherals; keypunch facilities were provided inside the computer room; scratch tapes were immediately available; and engineers were assigned to maintain the hardware.

As computers grew in capability and cost, the opportunity for individual control lessened. The price of user convenience became too high. A way was sought to increase the productivity of the computer. Uncorrelated service routines such as memory dumps, loaders, compilers and assemblers were brought together under one supervisory program. By the early sixties supervisory programs were gaining control of the operational environment of the computer, causing the activities once manually performed in the "hands-on" mode to now be performed "hands off." System utilization increased, and by superficial measurement, increased systems productivity claimed. However, user productivity suffered. Runs frequently were aborted due to an improper monitor control statement, user awareness of the event following hours after its occurrence. Large and frequent core dumps, whose processing consumed equipment and increased turnaround, were taken in order to trap in print much of what once had been observed by eye. These and similar events discouraged the installation manager because, in spite of a highly efficient operation, his stated goal—increased overall productivity—was missed.

Multi-programming and rotating machinery, i.e., drums, discs, etc., offered some relief to what had become an intolerable situation. Processing files from drums and discs reduced tape utilization, and improved channel usage between main and auxiliary storage. Multi-programming systems provided for a more efficient use of the processor to process data and to service peripherals. Multi-processing held out the promise of lower cost through peripheral sharing, processor backup, and abundant capacity for special processing requirements. Extremely high system utilization was achieved, but the user was still forced to negotiate with the computer in a "hands-off" mode. This meant (1) he was unable to address the system except by those devices resident within the computer room; (2) he was forced to predict and prepare for all possible events that could occur during the execution of his program; and (3) his employment of the computer was through a manual operations procedure not under his control.

THE UTILITY

Early form

The effort that surmounted this impeditive mode was the imaginative coupling of software, hardware and communications capabilities into systems designed to accommodate remote access and user interaction and to remove manual intervention. The card reader, printer, and keyboard, in becoming terminals, offered to bring the computer to the user. Simplified and well-designed control languages and file editors made access to processor and files more convenient and approachable. A teleprinter became an on-line keypunch as well as an input and output station; card readers and high speed printers were available when needed, and no one intervened. The user at his terminal was regaining the position of control he had enjoyed previously when he processed from within the computer room. A computer could now be utilized in a convenient and productive manner from wherever its many concurrent users were located.[1] The utility had its beginning.

Operational problems

Initially, this new operation mode was accompanied by new operational problems. Field maintenance, essential for terminal up-time, was poor. Existing tape library management was inappropriate because run request forms no longer existed. Operator console messages were frequent, untimely and ambiguous. Data communications were unreliable, undocumented and without established operational procedures. Central site hardware maintenance was difficult to schedule, because output remaining to be transmitted was resident to and controlled by the central processor. Errors were difficult to trap because of the urgency in continuing on-line communications. Recreating errors was impractical because the dynamic status of the environment at the time of failure was unknown. Changes to the monitor system were difficult to test because of the lack of experience about the environment in which they would exist. Preparation, distribution and user notification of changes was nightmarish.

Central site power was increasingly consumed by terminal devices whose individual throughput was less than the central site devices they replaced. Early experience demonstrated a dramatic mis-match between terminal speed, line speed and processor speed. The processor became output bound, precluding any computational activity or the acceptance of any additional input requests. The first action taken to alleviate this problem was to add additional storage capacity for I/O buffering. This solution did not accelerate the throughput of the system but merely delayed the point at which the system again became output bound, causing another calamity—delay in turnaround time. In addition, timely communication between the remote terminal operator and the central site operator, essential, for example, in order to suspend output to replenish printer paper, make forms changes, or correct terminal malfunction, was non-existent. Numerous tape mounts caused severe operator intervention, adding to central processor idle time. Queueing was primitive and a run requiring seconds of execution could find itself behind a half-hour run, and/or its output stacked behind three hours of printing. Adding to the upset was the absence of data security and privacy that contributed to lost and misdirected files.

Solutions

The introduction of communications subsystems solved some of these problems. The communications subsystem consisted of a small processor especially programmed to manage the communication between the central computer and the remote terminals. Data compression, site verification and operator communication were among the tasks well performed by this subsystem. Communications handling was thus removed from the central processor.

Separate queues were established in the central processor for tape and non-tape runs, as well as for long and short runs. Tapes were pre-mounted before run execution and short non-tape runs were executed on a regularly scheduled interrupt basis. I/O files were site code identified and terminal verification was established before data transmission began. Notice of changes to the system was included in the transmission to each terminal site. Hardware selection centered around the computational and the communications processors. The computational processor was selected according to maintainability, reliability, instruction repertoire, execution speed, and other historically accepted criteria. The communications processor, in addition to the above, was selected on its ability to easily handle multiple and dynamically switched priorities, to accommodate various speeds and dissimilar modems, as well as readily interface with multimedia mass storage devices.

Measurement

Instrumentation was installed to measure system productivity and to determine the profile of the most avid, and therefore, perhaps most productive user. Among the data collected were (1) execution time vs. output volume, (2) tape vs. no-tape runs, (3) average number of tapes per tape run, (4) terminal media by which runs were submitted, (5) time of day for heaviest traffic, (6) connect time vs. execution time, and many other criteria independent of and integrated with these.

The analysis of these data revealed that more efficient utilization of the system was achieved. There was less idle central processor time, faster response to more users, timely operator communication with the terminal site, and much improved tape procedures. The analysis also suggested that the configuration of a computing system could be optimally adapted to a given job mix. The utility's most productive user worked in an engineering discipline, demanded minimal input/output, was remote batch oriented, executed his own applications, and was substantially independent of operations personnel. This user regained hands-on posture, essentially because he was *autonomous* in his operating mode. From his terminal he initiated and resubmitted runs, made his own corrections at his own pace, and therefore bypassed the impeding nature of a manual operations procedure. He worked in his habitat and he shared with and solicited comments from those with whom he worked. His need for printed output, especially during debugging, diminished as did his need for punched card files.

Extending the service

Enthusiasm within the utility community grew as more operational problems were resolved. This enthusiasm attracted another class of users who had an opposite profile to the earlier class. This new user required numerous tape files, voluminous print files, forms changes, reprints, punched card output, and other services requiring manual intervention. This user executed runs that had a much higher ratio between I/O and processor time (e.g., 40:1 compared to 8:1), and he represented a group whose computing dollar expenditure was greater than 90 percent of the total expenditure of the computing community.

This business data processing user could only partially capitalize on the increased productivity offered by the computer utility because of the above stated characteristics of his work profile. The design legacy bequeathed his program, many of which could trace their roots back to the single function, manual step mode of unit record equipment, contributed significantly to this profile. Much of the manual processing flavor of these systems remained when conversion occurred. Operational compatability for some of these systems during their conversion from unit record machines to computers was, in fact, a major consideration. Furthermore, indeterminate output delay, due to a data communications failure, was sufficient cause to create a sense of real time urgency in payroll departments, and the possibility that company data could be compromised without detection was intolerable. Continuing investigation to understand these problems brought the utility user's profile into sharper focus. For example, a geologist programmer required multi-media output and the data he processed was, to him, urgent and private. Business programmers also did "compile-and-go" testing. Thus it became apparent that this was not the case of the engineering vs. business user, but rather the case of the *autonomous* vs. the *non-autonomous* user.

Work began in two areas to provide for the inclusion of the non-autonomous user. At the central site additional file processing schemes were installed. Print files were spilled over from drum to tape in background mode for subsequent low traffic distribution. Store and forward service was introduced permitting daytime transmission and nighttime execution of the run, or vice versa. Rotating machinery for additional buffering, as well as higher speed and full duplex lines for more rapid transmission, were added to the communications subsystem. Terminal site soft-ware was modified to facilitate additional operator communications and to service the equipment needed for local peripheral processing.

CURRENT STATUS

By now, computer installations operating in a utility mode employ several million miles of communications lines serving over 200,000 keyboard terminals, and over 20,000 high speed card reader/printer terminals. User productivity was the objective of the early installations. The growth in the number of computer utilities is attributable to that objective being met, to increased user sophistication, and to realized additional value by the installation manager.

Users continue to become more aware of the computer's value and less concerned with the computer. The utility provides a means by which the user can gain convenient access to the maximum contemporary computer power he requires. Company or department size may have little relation to the size of the computing problem at hand. With the commercial utility, a small firm has access to a resource similar to that available to the large corporation. With the in-house utility, all users within the organization have the opportunity to capitalize on the total computing resources available.

There are several values realized by the installation manager. First, there is the reduction in shake-down time of new systems software and a manpower savings due to centralized software maintenance. Centralized usage being at least equal to the sum of the usage of previously individual sites causes bugs to show up sooner, and when the bug is fixed, it is fixed for everyone simultaneously. Second, there is a low cost opportunity for expansion. Additional capacity can be added to a utility operation on a cost basis distributed throughout the user network. And, this expansion provides a low cost opportunity for back-up heretofore denied individual sites. In addition, there are savings in the overhead costs such as physical plant, hardware maintenance, site security, and parts inventories.

And, finally, there is the opportunity to focus resources on the challenges of the future. Among these are (1) low cost reliable data communications, (2) simpler user file processing procedures, (3) more efficient on-line mass storage, (4) pricing, (5) data privacy and security, and (6) effective I/O file processing.

SOLUTIONS IN PROCESS

Items (1), (2) and (3) of the preceding paragraph are receiving abundant attention. However, pricing remains as much a challenge to explain as to accomplish. Clearly, a utility user's job consumes time and space within the processor and the communications subsystem, space on peripheral storage devices, channel time to move data to and fro, communications lines and terminal devices. Further, there is the question of continuous operation and priority requests for critical needs. A multi-environment (user/processor) exists that precludes use of the historical time quantum and storage allocation pricing scheme. Determining the profile of the utility user and the effect this profile has on the total system is necessary in order to achieve efficient operations and precision pricing. A unit of cost based on user value discernible prior to execution and consistent within reasonable tolerance seems to be in the offing.[2]

Data privacy is in its own right an emerging industry being prodded into existence because of the sheer dollar value at risk. Corporation data and computer programs are assets with high determinable value. Schemes and procedures for the protection of these assets are, for many installations, non-existent and, often where present, inadequate. The installation manager faces a severe challenge in finding the procedure to achieve data privacy and security. Within this procedure must be the means to prevent accidental and deliberate acts that destroy or jeopardize the storage, processing and retrieval of data.[3] And this protection, this electronic lock[4] that may include cryptographic enciphering of files, must be installed without restricting or encumbering authorized user access. The security procedure must control physical access to the computer room, tape library and card file room, audit installation transactions and encourage a general security consciousness in installation personnel.

Speedy I/O handling is as critical as the processing of the data. Transmission queueing and intermediate storage of the terminal I/O files are among the significant factors contributing to the complexity of the developing I/O processing solution.

SUMMARY

As the past has demonstrated, solutions to processing problems will be found and, in turn, will provoke imaginative new ways to consume the power of the computer. This being the case, the installation manager will continue to face the decision of what operational mode his installation should employ. The utility makes access to contemporary and consummate computer technology simple and convenient, removes the geographical limitation of those working in cooperative efforts, and frees talent for creative activities by automating clerical procedures.

The economic value concomitant with these achievements, whether it be yield on the utility investment or quality of the final product resulting from the availability thereof, will be left to the determination of the reader.

The utility has emerged. As it matures it will move closer to the realization of its potential which is to bring the value of the computer to all who request it wherever and whenever they desire.

REFERENCES

1 F J CORBATO M M DAGGETT R C DALEY
An experimental time-sharing system
Proceedings of Spring Joint Computer Conference 21 Spartan Books Baltimore 1962 pp 335-344

2 M SEVCIK H WALDBURGER
Ein Verfahren zur quantitativen Berechnung der Computerleistung und -energie
Neue Technik Zurich Switzerland Nr A6 November 1970

3 J M CARROLL P M McLELLAND
Fast 'infinite-key' privacy transformation for resource-sharing systems
AFIPS Conference Proceedings Fall Joint Computer Conference Vol 37 pp 223-230 1970

4 D R WHITSON
Computer data privacy
Speech presented before National Secretaries Association Seminar Dallas Texas 1970

Installation management—The next ten years

by RONALD M. RUTLEDGE

Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

In this paper we will give our extrapolation of the exciting challenges facing installation managers in the coming years and how we expect to react to these challenges. We view the evolution to the present stage as having gone through three generations of installation management. The first generation was marked by the early computers with small memories and relatively crude input devices in which the operation of the computer was done mainly by the person using the computer. In this early generation the computer installation manager had the hardy pioneer spirit and knew most of his customers by first name. One of his main worries was how fast he could get the next model of the computer.

The second generation was marked by the introduction of tape operating systems and some primitive disc systems which required that the person operating the computer be very nimble and able to react quickly to the need for changing tapes so that computer utilization would not drop. In that generation the customer moved away from being in direct contact with the computer but there were still many customers who used the computer directly. Input and output were becoming a problem. The installation manager now had to start worrying about planning the layout of the I/O area carefully and was looking forward to the coming generation for order of magnitude improvements.

In the third generation we found the introduction of multiprogramming facilities through the use of primitive drums and improved disc facilities. And here, as in the second generation, the need for competent operation of the equipment became of increasing importance. One then had many jobs requiring tapes and discs running at the same time. The I/O problem increased even further and the great hope of eliminating it through terminal access was dismally smashed. The utilization of remote job entry stations did finally ease the problem in the latter part of this generation. By using remote job entry stations, most customers no longer required their own computer nearby to get adequate computing service. In this generation the installation manager had to face up to the full problems of being a production specialist, a resource accountant, a businessman and a technocrat.

In what we now view as the fourth and present generation at Carnegie-Mellon University the distinction between the generations becomes hazier. We will list the main characteristics of this generation. Awareness that computers cost large sums of money and that poor utilization is not uncommon here become permanent issues with the economic decline. The possibility of inexpensive mass secondary storage is becoming very real and in some cases is almost here. At times our complex systems appear to be working stably. Computer installations are no longer simple one machine, one vendor sites.

Telecommunication is now as big a problem as the computers and people are asking what they are getting for their dollar rather than how fast they can get the next piece of equipment.

The fifth generation we project will be characterized by low cost mass secondary storage, so inexpensive and efficient that tape drives will become uneconomical to use in many cases. Telecommunications will be reliable and economical, so that we will not hesitate to use a computer at the opposite end of the country. Time-sharing systems will actually work reliably. No one will take note of using the system with several hundred people on it. Computer installations will become rather amorphous entities consisting of complex inner connections of many different CPUs' memories, and various storage devices.

In the fifth generation, management information systems will become paramount. Low-cost mass storage will make implementation easy and the interconnections of networks will make distributive data bases a reality. The challenges in the coming years are put into the reference framework of cost effectiveness. Cost effectiveness is the main theme of our management's interest and it should be and probably will be the main theme for the foreseeable future.

By meeting some of the challenges mentioned, Carnegie-Mellon University (CMU) was able to cut its Computation Center costs by 40 percent and at the same time improve service, deliver more computing power, and increase customer satisfaction in a short period of a few months. Some details of this accomplishment have been covered elsewhere.[1] In our experience, innovation and implementation of automation is not an easy straightforward process, but the challenges can be met.

PERSONNEL AND ORGANIZATION

One of the largest voids in today's information processing technology is the lack of qualified educators to train the personnel we need. The ACM Curriculum Committee on Computer Education has presented reports[2,3] which will probably form the foundation of many curricula for training our future personnel.

The increasing complexity of the computer installation requires a shift to more technically competent personnel for daily operations. Unfortunately this shift in general has been too slow and many present installations suffer with reduced productivity due to less than optimally trained personnel.

Several years ago there were studies[4–7] which revealed that the difference in the productivity of programmers can be an order of magnitude.*

Unfortunately, evaluating programmers[3] remains in a state of witchcraft. We cannot hire enough experienced programmers, and we are willing to train or hire inexperienced persons. Hopefully, by evaluating them over a period of employment, we can get a few very good ones from the crop. CMU receives big dividends in minimizing the actual number of personnel while maximizing on the competence of individuals. In other words, the incremental cost of properly compensating the excellent programmer is well worth the price if we are lucky enough to have him.

The actual organization of a computation center is not so important since the organization should reflect an attempt to optimize the contributions of the individuals and inter-personal relationships with respect to the objective function of the group. Therefore, one would reasonably expect the organization to change in time as the different members of the team come and go. Though the usual organizations are somewhat reluctant to be highly dynamic, the rapid change in the advancement of information processing technology presently allows one to have a flexible organization without having the appearance of a continual card shuffle.

At CMU the shift to better qualified personnel has taken the route of the control center concept[1] with the elimination of the usual operations groups. The computation center staff sections are hardware, information, and software systems and with few exceptions every professional takes his turn being in charge of the control center. The programmer in charge of the control center may have as many assistants as the load requires. This approach is parallel with more automated operations requiring more technical skill and has significantly reduced operating costs. Better service and a more stable system result because of the increased technical ability of the personnel in charge to answer inquiries and react to contingencies.

Personnel are the only important resource. Whatever machine we have today will certainly be outdated in five years whereas our personnel will be more valuable in five years if we give them the proper motivation to keep abreast of our challenging field.

THE EXECUTIVE GAP

Most computer installations are a part of a greater organization which the installation serves. With the increasing utilization of computing resources the importance of the installation to the parent organization is increasing. The rising costs of these computing resources to the parent have forced many organizations to recognize the need for an executive position with the responsibility for coordinating and integrating the organization's computing resources. This executive must be technically and administratively competent.

In most organizations today there is not an executive position with adequate technical competence for the magnitude of importance of the organization's computing resources. One or more installation's managers have been given default technical responsibility without proper administrative authority for effectively handling the responsibility. One common symptom of this executive gap is separate computer installations for "administrative" and "scientific" computing. The managers and their organizations who recognize and react to this common gap in their executive structure have much to gain in better utilization and reduced total costs.

* These studies were attempts to evaluate the relative merits of time-sharing versus batch processing as a means of programming. Although the intended results were inconclusive, the wide range in programmer productivities was repeatedly demonstrated.

Until the executive gap is filled the installation managers must understand the requirements of the organizations they serve as well as keep abreast of computer technology if their service is to be efficient. This requirement of having an executive position which must be alert to developments in the rapidly changing computer technology is one of the toughest challenges to be met if our organizations are to receive the proper benefits of our technology.

CONFIGURATIONS

Will the small computer installation disappear and be absorbed by the giant emerging ones or will a small computer installation have a place in the future? The answer to this question obviously depends on the circumstances. Just as a small switchboard has a place in our present day telephone system, there are many circumstances where the small switchboard doesn't belong. In our projection it will be more than ten years before the small computer installation is really threatened in the same way as a small telephone switchboard. Certainly the installations that will exist over the next few years will become more amorphous entities rather than one single CPU or one single manufacturer. Reliability will be markedly improved[9] through new circuitry,[10] improved hardware technology,[11] and better software.[12,13] Emerging large mass storage systems[14,15] will permit a greater degree of automation and accommodate the developing management information systems. The general population is beginning to understand the basic principles[16] of machine configurations. We now have our first book characterizing in a systematic way the various computer structures.[17] The minicomputers won't replace the large CPU's but they will certainly supplement and complement them. Every large CPU probably will have numerous minis around in various functions for which the larger CPU is less cost effective.[18]

NETWORKS

Computer networks[19] are another area in which the small computer installation is threatened by replacement with a remote job entry facility or a few terminals. This will eventually happen, but we certainly do not think it will be in the next few years. It is, however, definitely a possibility in five to ten years. Communication[20,21] problems have held the development of networks back many years. However, the great thrust provided by the underwriting of the ARPA Network has provided the enthusiasm for exploring the present potential of networks.[27]

Actually there will be two forms of networks, the local network and nonlocal or long distance network. There are many advantages to local networks which haven't been properly realized by the installation manager. Many installations have various computers with duplicated hardware and software facilities.

The linking of local computers and the reduction of redundant facilities will result in the elimination of duplicate costs and also in increasing cost effectiveness because jobs will be easy to run on the appropriate machine. This was successfully accomplished recently at CMU with sizable cost saving (capital and operational), increased effectiveness, and added functional capability for the customers which formerly used each system as an independent entity.[1]

Also the reliability of the total installation will be improved since one can transfer a job more easily from one machine to another. The non-local or long distance networks will satisfy peak load demands and actually permit installations to run at a greater average capacity because they will not have to have an excess capacity to satisfy peak loads. The result will also be an increased cost effectiveness. The non-local networks will allow each installation to give one stop service to a broad spectrum of functions which will be available through the networks to which it is connected.

RESOURCE ACCOUNTING AND CONTROL

The measurement of computer systems has now come of age with the first annual workshop on software system performance evaluation held last year[28] and installations are developing their own means of implementing primitive measurement facilities. In the coming years manufacturers may actually supply measurement facilities as parts of the standard system. We should not have to buy a computer as though it were a car without a speedometer or gas gauge or odometer. The question of pricing,[29] cost accounting, and the administration of computer resources* is now considered an integral part of the resource accounting and control schemes.[30,31] The marketplace will demand the facilities to measure and record utilization. Whether one should lease or buy and how and why one should write off purchased equipment is a commonly discussed issue. The marketplace will discover that

* With proper accounting and pricing sometime during the coming years we will stop referring to our customers as 'users.' The label 'user' has adverse psychological effects for the 'user' and the ones 'used.'

purchasing equipment and using some sort of declining balance depreciation is the proper and logical way to cost account for equipment resources of a changing technology.

HARDWARE PROJECTS AND MAINTENANCE

This area is becoming a challenge due to the increased in-house capability of installations not only to maintain but also to fabricate and enhance their equipment. The recent emergence of the 'Brand X' peripherals has created a problem in many installations with multi-vendors. Now the installation requires at least one person competent in hardware logic to minimize the potential conflicts between various vendors. For those with the capability, a potential savings of some magnitude exists for in-house maintenance of equipment.[1] But for those who do not prefer this route there are third party vendors who will furnish maintenance for various manufacturers' equipment.

SOFTWARE PROJECTS AND MAINTENANCE

Software still continues to be one of the biggest challenges. The Third ACM Symposium on Operating Systems[32] last year gave us the impression that perhaps operating systems were now being comprehended.

In general we are beginning to see logical structures and approaches to putting systems together. Three annual conferences have been held on software engineering. Their accompanying proceedings[33–35] project that an emerging profession is developing in software engineering. In the next few years we will even be getting the handbooks or cookbooks which will tell our software engineers what pieces to use for the type of system we want to put together. The education, training, and development of programmers and operators is still somewhat of a controversial subject.[36] Some curricula are becoming standardized and our first book on management of computer programming was recently published.[37] Perhaps people have not learned that we still have to develop the art of managing a large software project. It has been our experience that the project employing more than three programmers is likely to have serious problems without careful structuring of the project[38] and software.[39] With the continual unbundling that is occurring perhaps the vendor software groups will have to stand on their own or face up to competition from the outside. Then the quality of the vendor software will be improved or the purchaser will not opt to take it. This will hopefully create more competition where installations can go to third party for maintenance of their standard operating system and have more options to buy software packages built to order for them.

CONTRACTS AND LAW

Contracts[40] and law unfortunately tend not to be recognized as the challenge they are, although many of us have recognized this the hard way. Monopolistic tendencies of the various competitors in the marketplace require the installation manager to have an understanding of the Antitrust Laws.[41] The field of law is also now becoming aware[42] of the particular requirements of the computing technology.

STANDARDS

An old and difficult challenge is still with us. The emergence of the networks has shown that if the networks are going to work, which they surely are, people will be forced to use standards.[43] The developing of microprogramming[44] capabilities hopefully will allow not only better standards but also allow a standard operating system which would run on most computers and still allow the computer to have its own unique capabilities.

COMPETITION IN THE MARKETPLACE

Do you really have competition in the marketplace? With the folding of RCA Commercial Computer Division and with GE electing to get out of the field, some think that competition is decreasing rapidly to zero. Of the many and various extreme views on this subject one of the more moderate was expressed by Larsen.[45] He said, "Hurting IBM is easy; solving national problems is tough and difficult. A good solution requires all of the business and technical skill at the industry's disposal. It also requires that we care—not about IBM, but about the United States, if we are to keep the United States in its lead of the computer technology." Larsen, unfortunately, did not cover the case of an international monopoly.

The real question is whether or not our workable but competitive system gives the marketplace a fairly priced and up-to-date product. If one company spends more on research and development than most of its "competitors" gross[46] is it possible to compete rather than just trail behind and subsist on leftovers? Could Fortran and OS/360 have survived in a free and knowledgeable marketplace? How many installations have computing contracts tailored to their requirements and explicitly hold their vendors liable for failure to deliver as promised rather than the "standard" contract loaded with the "come hell or high water the installation will pay the vendor" clauses? Does the installation have the engineering specifications for equipment worth millions? What can the installation do if its equipment is not maintained properly? Go to another company for maintenance? We must decide if the answers to these and many other such questions are satisfactory. If the answers are not satisfactory then one of the most difficult and important challenges facing us is what to do to obtain a marketplace where the competition benefits the customers.[47]

The motivation behind present day monopolies is not usually the hungry greed of the communist's stereotyped capitalist, but rather nothing more than the desire for the easy life (i.e., a secure well-paying job). To quote Judge Wyzanski,[48] "Some truth lurks in the cynical remark that not high profit but a quiet life is the chief reward of monopoly power. And even if a particular enterprise seeks growth and not repose, an increased rate in the growth of ideas does not follow from an increased concentration of power." This understandable motivation permeates the bureaucracy of the monopolistic organizations with the result that the monopoly power is not overtly exercised by any one individual but by a collective effect sometimes expressed as, "It is our policy", "We must treat everybody the same", or "We can't make exceptions". The salesman representing the monopoly, though benefiting from the monopoly, is quite helpless at combating his company's policy. This indirect collective effect can only be effectively changed by collective effort. Why can't the customer groups make their own standard contracts?

The Justice Department has historically acted very slowly on monopoly power. The large organizations with power to act do not want to "cast the first stone." The small installation cannot afford the time and money, and may times is justifiably afraid to fight. A giant organization is a fierce and ruthless opponent[49] but our obligation to society demands that if we decide detrimental monopolies exist, then we must face the challenges.

MANAGEMENT INFORMATION SYSTEMS

Corporations and private communities are recognizing the need for and the potential benefits of being able to make more informed decisions through the use of computers. Unfortunately these areas perhaps have been the hardest to penetrate due to the classical conservatism of the management[50] which will use these systems. Operating systems have not been outstanding successes in delivering past promises, but now that operating systems have become reliable, many different management information systems exist. Businesses are demanding the implementation of systems.[51,52] In fact the Government[53] feels they are absolutely essential to the future of the congressional operation.

INNOVATION AND AUTOMATION

Innovative operations through automation should be the theme of the installation manager. He has at his disposal the machines which represent enormous advances made in automated processes and which, by means of information systems, have furnished us the ability to make informed decisions.

Why hasn't the installation been able to exemplify the pinnacle of automation and innovation by allowing the machine to control the routine operations? With our society's becoming suspicious of computers[54] and their encroachment upon privacy, installation managers will have to be innovative to impress society[55] that the powerful facilities are in competent hands. In the post-industrial[56] era, with information doubling every 12 years,[57] the challenge of keeping track of one's own installation will continue and should be met with our own technology.

SUMMARY

Innovation and automation: with this as the theme of our coming years, the installation manager should set the goal of making his installation the pinnacle of innovation and automation. He will have to be prepared to justify to management not only that his present equipment is cost effective but that what he proposes to get in the future will improve the installation's cost effectiveness. Management is forced to recognize the large cost that information processing technology is now starting to represent with respect to the organizations' total budget. We are all very lucky to be a part of this exciting era but we must not make the mistake of leaning so much on our lessons of the past to extrapolate our actions into a very innovative and challenging era which may be totally different from our past experience.

ACKNOWLEDGMENTS

Although the director prepared this paper, the Computation Center Staff originated many of the ideas

presented here, and their exceptional competence and exemplary dedication made possible this exploration and the actual utilization of some of these ideas.

REFERENCES

1 H B ROWELL JR R M RUTLEDGE E SCHATZ
Management control
Proceedings EDUCOM Spring Conference 1971

2 *ACM curriculum committee on computer education for management education related to the use of computers in organizations*
Communi:ations of the ACM Vol 14 No 9 pp 573-588 1971

3 J L McKENNEY F M TUNGE
The state of computer oriented curricula in businss schools 1970
Communi:ations of the ACM Vol 14 No 7 pp 443-448 1971

4 M M GOLD
Time-sharing and batch processing: An experimental comparison of their values in a problem-solving situation
Communications of the ACM Vol 12 No 5 pp 249-259 1969

5 H SACKMAN W J ERIKSON E E GRANT
Exploratory experimental studies comparing online and offline programming performance
Communications of the ACM Vol 11 No 1 pp 3-11 1968

6 L B SMITH
A comparison of batch processing and instant turnaround
Communications of the ACM Vol 10 No 8 pp 495-500 1967

7 M SCHATZOFF R TSAO R WIIG
An experimental comparison of time-sharing and batch processing
Communications of the ACM Vol 10 No 5 pp 261-264 1967

8 R ARMSTRONG R BERGER L FERRETTI J NICKERSON J M PALERMO
A panel session—People—Selecting and evaluating the computer workforce
Proceedings ACM 70 pp 21-27 ACM 1971

9 Special Issue on Fault-Tolerant Computing IEEE Transactions on Computers Vol C-20 No 11

10 J S GOLD
On automatic testing of online real time systems
Proceedings SJCC 1971

11 T KURODA T C BUSH
Multiband automatic test equipment—A computer controlled checkout system
Proceedings SJCC 1971

12 S A SZYGENDA
Coding techniques for failure recovery in a distribution modular memory organization
Proceedings SJCC 1971

13 D L DROULETTE
Recovery through programming system/360-system/370
Proceedings SJCC 1971

14 R B GENTILE J R LUCAS JR
The TABLON mass storage network
Proceedings SJCC 1971

15 M SAKAGUCHI N NISHIDA T NEMOTO
A new associative memory system utilizing holography
IEEE Transactions on Computers Vol C-19 No 12 pp 1174-1180 1970

16 M J DURAO JR
Finding happiness in . . . extended core
Datamation Vol 17 No 16 pp 32-34 August 15 1971

17 C G BELL A NEWELL
Computer structures: Reading and examples
McGraw-Hill 1971

18 C G BELL A NEWELL F P BROOK JR D B G EDWARDS A KAY
A panel session—Computer structure—Past, present and future
Proceedings FJCC 1971

19 C G BELL A N HABERMANN J McCREDIE R M RUTLEDGE W WULF
Computer networks
Computer Vol 3 No 5 pp 13-23 1970

20 R G NYSTROM L M PASCHALL J M RICHARDSON W J SULLIVAN
A panel session—Communications: Challenges of the seventies
Proceedings ACM 70 pp 34-43 ACM 1971

21 W M ELLINGHAUS J H ENGEL D F FOSTER J K LADY R H McCONNELL W G McGOWAN
A panel session—Communications: Data transmission
Proceedings ACM 70 pp 44-58 ACM 1971

22 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
Proceedings SJCC 1970

23 F E HEART R E KAHN S M ORNSTEIN W R CROWTHER D C WALDEN
The interface message processor for the ARPA computer network
Proceedings SJCC 1970

24 L KLEINROCK
Analytic and simulation methods in computer network design
Proceedings SJCC 1970

25 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
Proceedings SJCC 1970

26 C S CARR S D CROCKER V G CERF
HOST-HOST communication protocol in the ARPA network
Proceedings SJCC 1970

27 R M RUTLEDGE A L VAREHA L C VARIAN A H WEISS S F SEROUSSI J F JAFFE M A K ANGEL
An interactive network of time-sharing computers
Proceedings 1969 ACM National Conference

28 Proceedings ACM SIGOPS Workshop on System Performance Evaluation Harvard University 1971

29 N R NEILSEN
The allocation of computer resources—Is pricing the answer?
Communications of the ACM Vol 13 No 8 pp 467-474 1970

30 R HARRIS J NOERR T W RINEHART C E SKIDMORE
A panel session—Management: Administration of computer system resources
Proceedings ACM 70 pp 351-353 1971

31 L L SELWYN
Computer resource accounting in a time-sharing environment
Proceedings SJCC 1970

32 Proceedings ACM Third Symposium of Operating Systems Principles Standford University 1971

33 P NAUR B RANDELL
Software engineering
Report on a Conference Sponsored by the NATO Science Comm'ttee Garmisch Germany October 1968

34 J N BUXTON B RANDELL
Software engineering techniques
Report on a Conference Sponsored by the NATO Science Committee Rome Italy October 1969

35 J T TOU
Software engineering
Vol 1 and 2 Academic Press 1970 1971

36 D ABE J D BENENATI H CADOW R S McKENTY
A panel session—Education: Training and development of programmers and operators
Proceedings ACM 70 pp 89-95 1971

37 G F WEINWURM
On the management of computer programming
Auerbach 1970

38 F T BAKER
Chief programmer team
Proceeding of SHARE XXXVII Vol 1 pp 240-251 New York August 1971

39 E W DIJKSTRA
The structure of the The multiprogramming system
Communications of the ACM Vol 11 No 5 pp 341-346 1968

40 L P SIMPSON
Handbook of the law of contracts
West Publishing St Paul 1954

41 J G VAN CISE
Understanding the antitrust laws
Practising Law Institute New York 1966

42 R N FREED
Materials and cases on computers and law
Roy N Freed Esq Boston 1971

43 N R NIELSEN
The merit of regional computing networks
Communications of the ACM Vol 14 No 5 pp 319-326 1971

44 *Special issue on microprogramming*
IEEE Transactions on Computers Vol C-20 No 7 1971

45 G H LARSEN
DP monopoly hurting the US computer world
p 10 November 17 1971

46 *International business machines*
Forbes Vol 108 No 5 pp 18-22 September 1 1971

47 *Antitrust: Storm signals*
Forbes Vol 108 No 9 p 23 Nov 1 1971

48 C WYZANSKI
United States vs. United Shoe Machinery Corporation
110F Supp 295 D Mass 1953 aff'd 347 US 521 1954

49 W RODGERS
Think
Stein and Day New York 1969

50 E BERKELEY H GOLUB C J GRAYSON P NORDEN J TENNANT
A panel session-management: Management education in data processing
Proceedings ACM 70 pp 344-350 1971

51 H KRIEBEL
Summary of the management sessions
Proceedings ACM 70 p 343 1971

52 R A GILBERT J T GILMORE T W OLLE F G WITHINGTON
A panel session-management: Management information systems
Proceed'ngs ACM 70 pp 354-358 1971

53 J E HUGHES E J MAHONEY P MEDOW E R TURNER
A panel session-government: Computers in government
Proceedings ACM 70 pp 174-191 1971

54 P KAMNITZER N F KRISTY J McLEOD E W PAXTON R WEINBERG
A panel session—Computers and the problems of society
Proceedings FJCC 1971

55 B W BOEHM D COHEN B NAUS N R NIELSEN E B PARKER
A panel session—Planning community information utilities
Proceedings FJCC 1971

56 H R J GROSCH V C HARE C P LECHT H PERLMUTTER J M STEWART
A panel session—Management: Management in the post-industrial era
Proceedings ACM 70 pp 359-365 1971

57 L G BRYAN M S S MONTON K L ZINN
A panel session—Education: Recommendations for action
Proceedings ACM 70 pp 71-79 1971

A set of goals and approaches for education in computer science

by SAUL AMAREL

Rutgers University
New Brunswick, New Jersey

INTRODUCTION

In order to evaluate and plan educational programs in computer science it is important to have a clear conception of the internal structure and content of the discipline, and also of its relationships with other disciplines. It is also important to have a clear set of goals, values, and priorities in the light of which given programs can be evaluated, and new approaches can be developed.

In this paper I will discuss a specific set of goals and approaches for educational planning in computer science. Much of our planning at Rutgers reflects the viewpoints that I am presenting here.

THE STRUCTURE OF COMPUTER SCIENCE

In Reference 1 I have presented a framework for curriculum planning in computer science which combines a global view of the discipline and a local view. The global view is concerned with the kinds of objects, phenomena, and concepts that form the domain of discourse in computer science, and also with the pattern of relationships between this domain and other domains of study. The local view focuses on a structured description of the knowledge, problems, and activities within the discipline.

In developing the local view of the field, I find it useful to distinguish between two broad categories of activities: (1) activities concerned with problems, methods of solution and programming; and (2) activities concerned with representations, languages, schemes of processing, and system designs. They correspond roughly to *computer applications* and to *systems*. Within each of these two categories there is a broad spectrum of types of activities: development of formal systems and formulation of theories; search for fundamental principles; invention and exploration of new schemes; design and construction of new systems; and experimentation with existing systems. The range extends from purely theoretical work to practical work within the state of the art.

Theory and practice

I think that it is important to maintain a view of the field which helps to *minimize the distance between theoretical and practical work.* Computer Science is concerned with analysis and synthesis aspects of various types of computer applications and systems. A continuous interaction between theoretical and design/experimental work is needed for a vigorous rate of progress in the field. Theoretical work in a rich applied environment is likely to receive stimulation along lines of 'relevant' models and theories. Design and experimental work in a good theoretical environment promises to lead to more efficient and powerful systems and applications.

An important reason for maintaining a strong coupling between theory and practice in academic computer science is our responsibility for guiding the growth of the discipline toward the dual goals of intellectual depth and coherence, and also of pragmatic significance for the real world of computing. Because of the phenomenal growth of the computer field, we are witnessing a rapid accumulation of vast amounts of unstructured information: specifications of many diverse hardware and software systems, records of a variety of operating experiences, discussions of different principles, techniques, and empirical advice, and many fragments of knowledge only weakly related to each other. In order to be able to build on top of what has been done in the past, and in order to understand and effectively use new developments, we must plan computer science programs in accordance with the following: (a) exposure to what is being done in the real world of computing, (b) study of ways for bringing order into what is being done, through an active search for unifying principles, general methods and theories, and (c) exploration of new computer designs and applications in the light of new concepts and theories.

Clearly any structural description of the subject matter of computer science reflects certain points of view and values about what we ought to teach and how. By not using a separate category for 'Theory' or 'Formal Systems' in a broad structuring of the field I am injecting a specific viewpoint into curriculum planning which favors 'rapprochement' between theory and practice in computer science programs.

Systems and applications

Our academic planning at Rutgers is based on further substructuring of the two categories 'Systems' and 'Applications' into the four parts 'Hardware Systems' 'Software Systems', 'Numerical Applications', and 'Nonnumerical Applications'.[2,3,4] The 'Hardware Systems' part covers mainly machine organization, digital subsystems, operating systems, and design processes. The emphasis in the 'Software Systems' part is on machine level representation and programming, operating systems and language systems. The 'Numerical Applications' area is mainly concerned with problems and procedures where numerical data are dominant, such as problems in numerical analysis and in statistical processing, optimization and mathematical programming, and certain classes of modeling and simulation problems. The 'Nonnumerical Applications' area is primarily concerned with processes involving nonnumerical data. Examples of such data are: representations of problems, situations and programs; symbolic expressions; language text; abstract structures; and graphic objects. In addition, each of the application oriented areas is concerned with relevant studies in high level languages, schemes for representing problems, data bases and procedures, and related theoretical subjects.

Each of the four parts described above provides a basis for a "topic of study", which may be chosen by a student as an area of concentration. The partition into the four areas reflects an attitude to curricular design which is widely accepted at present. It has a fairly sound conceptual basis, and it leads to an implementable program which can now be supported by considerable experience and a growing literature. The conceptual basis is essentially *reductionist*: computer systems are decomposed in terms of different types of structural components (structural properties determine possible modes of computation), and computer applications are differentiated in accordance with major types of data structures (properties of data structures determine types of information processes). This reductionist attitude will probably change in the next few years, as it will become increasingly desirable (and feasible) to treat hardware and software designs in a unified manner, and as the emphasis in application areas will be on broad problem types where both numerical and nonnumerical processes will play essential roles (e.g., modeling and decision making in medicine, large design and optimization problems). The changes are likely to come first in advanced parts of graduate programs, and they will gradually move toward undergraduate programs. I expect the changes to be relatively slow, not only because of the conceptual issues involved in restructuring the field, but also because of the natural inertia of educational systems where required courses, supporting texts, and other relevant resources, cannot change overnight.

A structured description of the field is an important input for planning an educational program, and it already reflects a specific set of viewpoints and decisions on priorities. We need, in addition, further clarification of priorities in order to determine (a) the relative emphasis that the program should place on different parts of the field, and (b) the way in which individual students should be guided through the program.

Our emphasis at Rutgers is on the *relationship and mutual impact of application areas and computer systems.* This implies a fairly balanced set of activities in applications and systems, with major effort in problem solving methods, computer procedures, languages, systems software, and associated theories. This approach is consistent with our view that studies of computation mechanisms and system designs should grow in an environment which is rich with significant and challenging computer applications.

COMPUTER APPLICATIONS

I believe that serious work on new types of computer applications is essential for the healthy development of computer science. The challenges presented by new applications and the constructive attempts to meet them will continue to be key factors in the evolution of computer systems and in the conceptual maturation of the field. Work on new applications will lead to increasing collaborative ties with people in other disciplines; this will increase the 'surface of contact' between computer science and other areas, to the benefit of all involved. In this context, computer science could provide an academic home for certain important activities associated with *applied mathematics*: the formulation and study of models, the development of methodologies for bridging the gap between a real life problem and its representation within a system wherein the problem can be solved, and the study of broad approaches to problem solving.

In our programs at Rutgers, computer applications are studied at various levels. A freshman-level introduction to computing is oriented to hands-on experience with computer problem solving, using BASIC on a time sharing mode. Sophomore-level courses in computer problem solving focus on a variety of problem types (numerical and nonnumerical) and on the use of several high-level languages—FORTRAN, PL/I, SNOBOL. Thus computer languages and their features are introduced in the context of their *use* for formulating computer procedures of different types in a variety of problem areas. The study of computer organization, machine level programming, and systems software follows the courses where problem solving is the dominant theme. Thus, a student is first exposed to *what* kinds of things computers do and how can they be *used*, and then he is led to consider *how* computer systems operate, how they are structured and how they can be *designed*.

The study of computer applications continues in our upper level undergraduate classes with courses in numerical methods, data processing methods and nonnumerical algorithms. At the graduate level, two introductory (and required) courses on numerical analysis and nonnumerical algorithms are followed by a collection of courses and seminars which explore in different degrees of depth a variety of application areas, procedures, and approaches to problem solving.

Recently we established an (NIH supported) research resource on *computers in biomedicine*. This provides a focus for collaborative activities with people in medicine, bioengineering, psychology and ecology on a broad range of problems that involve the use of computers in biomedical inquiry. This type of research activity is an extremely important environment for advanced graduate work and for faculty research in various aspects of advanced computer applications.

Nonnumerical applications

The study of nonnumerical applications is of central importance in a computer science program. It provides insight into the broad notion of computers as symbol manipulators, and it permits a clear view of many key information processing concepts that become obscured when presented only in the context of mathematical algorithms for numerical problems. Also, the domain of nonnumerical applications is extremely large, and it contains a great variety of significant problems in the field. In addition to the broad spectrum of problems from elementary data processing to complex decision and problem solving in artificial intelligence, the entire area of systems software (languages and operating systems) is mainly founded on nonnumerical processes and techniques.

Artificial intelligence

Work in artificial intelligence problems provides an excellent opportunity for the study of nonnumerical processes and for the clarification of certain basic issues in software design. It is important to work toward the development of closer links between artificial intelligence studies and other studies in the mainstream of computer applications and of systems design. I believe that this will be of benefit to computer science education, and in general, it will stimulate new, significant developments in the computer field.

It is becoming increasingly clear that there is no sharp dividing line between the problem solving procedures of artificial intelligence and the procedures of today's 'conventional' software.[5] It is useful to think that computer procedures lie in some sort of continuum, and each procedure in the continuum is characterized by the relative amount of systematic versus heuristic knowledge available to it, as well as the degree to which this knowledge can be efficiently exploited by the procedure for the solution of specific problems in its domain. At the one end of the continuum, where systematic knowledge is dominant and its mode of utilization is highly efficient, we have the customized procedures and the formally validated algorithms of the well developed computer application areas. At the other end, where amount of systematic knowledge is low, we have the general, flexible, goal directed procedures of artificial intelligence, where a set of relatively weak problem-specific principles are combined with heuristic methods for the control of processes that search for solutions. In between, we have the bulk of procedures that run on computers at present—including procedures that manipulate computer languages, and those that control computer operations.

The notion of a procedure continuum, where the emphasis is on types of information processes and on types of knowledge associated with them, promises to provide a fruitful conceptual framework for education in computer science. It would be interesting to explore it further in the context of future curriculum designs.

Numerical applications

Numerical analysis and several other subjects in the general area of numerical applications are among the best developed in the field in terms of systematic knowledge available and theoretical treatment of algorithms. I think that these subjects have an important place in a computer science curriculum, not only because of their great significance for many computer applications, but also because they provide models of 'high quality' procedures and of successful theoretical approaches to their study. Such models are needed for progress in other, less developed areas of the field.

INTERACTION WITH MATHEMATICS

The study of numerical applications is closely related to mathematical studies. A strong mathematical background (especially in calculus and linear algebra) is needed for work in most numerical applications. Conversely, numerical applications are in themselves significant topics of study in mathematics.

The question of *mathematical preparation* for work in computer science must receive careful consideration in the development of computer science programs. Theoretical work in computer science relies on several branches of mathematics and logic. In addition to the formal background needed for numerical applications, various topics in abstract algebras, in logic and foundations, in combinatorial analysis, in graph theory and in probability theory are needed for work in well developed areas of nonnumerical applications and of computer system design.

A problem aiises when we consider the training of people who are oriented to the bulk of design and application activities in the field. For the most, these activities are in areas where relevant theoretical knowledge is poor. There is considerable consensus among computer science educators that people working in these areas must have at least a certain level of 'mathematical maturity.' This does not imply necessarily the knowledge of specific mathematical facts and methods. What is meant usually is training in disciplined modes of thought, in the manipulation of abstractions, and in the development and use of methods that can apply in a great variety of situations. To obtain this type of training, we have several approaches: (a) provide sufficient exposure to offerings in regular mathematics curriculi, (b) restructure some mathematical offerings to attain within a limited time, both the 'maturity' objective and the objective of education in mathematical topics of special relevance to work in the computer field, and (c) develop computer-based courses that address themselves directly to the underlying cognitive skills implied by 'mathematical maturity,' without using conventional mathematical material. These approaches are not mutually exclusive. I believe that it is an important challenge for computer science education to develop the approach (c). In parallel, I would place considerable stress on the approach (b). I am basing this on the conviction that even those who graduate at present from a computer science program and expect to take a programming job which does not require an immediate specific knowledge of mathematics, must be equipped against rapid obsolescence in a field which is extremely dynamic and which requires continuous learning and adaptation to new concepts and techniques. As theoretical knowledge grows in their area of activity, they must be prepared to understand it and to use it. Formal approaches will continue to develop in computer application areas; they are likely to penetrate more and more in systems design areas as concern with quality and *efficiency of designs* will continue to grow, and as methods of system analysis and evaluation will become increasingly available.

Wallace Feurzeig and his colleagues at Bolt Beranek and Newman have started to design courses where computer programming is being used as a key to the development of certain basic mathematical ideas and of skills for abstract reasoning.[6] This work has been oriented so far to pre-college instruction. It would be extremely interesting to develop courses of this type for introductory college instruction. In addition to their effectiveness for the development of 'mathematical maturity,' such courses would be exceptionally well suited for the introduction of computers to a large number of college students who, because of their 'fear of math,' hesitate to take introductory computing courses in their present form (which is commonly biased toward mathematical calculations).

INTRODUCTION TO COMPUTING AND TO GENERAL EDUCATION ABOUT COMPUTERS

Computer science departments have an important responsibility to provide broad *introductory computing courses* to the entire College population. In order to reach a very wide group of students, with a great possible variety of backgrounds and orientations, it is important to develop courses with no special mathematics requirements and with the main emphasis on the essentials of information processing and on hands-on problem solving experience with computers. The design of such courses is of considerable value for computer science as a new discipline, since it forces attention on fundamental concepts in the field and on imaginative ways of communicating them.

One of the most promising areas for the use of CAI in a college curriculum is in introductory courses in computer science and in mathematics. It would be highly desirable to stimulate the development of such computer-based courses in computer science departments; this would be a valuable activity for both faculty and students.

In our program at Rutgers we have had some experience with an introductory course where the emphasis was on uses of computers and on their *impact on society*. We found that students at the freshman level were much more interested in what computers are and how they can be used, rather than in general questions about the impact of computers on society and the broad significance of a growing computer culture on people. I think that it is profitable and important to discuss these issues in a senior seminar; at that level, students have more knowledge (of computers and of other things), and more maturity and motivation to study relationships between computers, man, and society.

SUGGESTED APPROACHES TO LONG RANGE EDUCATIONAL PLANNING IN COMPUTER SCIENCE

The question of *educational breadth* versus *specialization* is central to long range academic planning in computer science. As the computer field grows, there is an increasing demand for in-depth training of a variety of specialists. If the rate of generation of general principles and methods in the field will continue to lag behind the rate of production of specific computer systems and specific application packages, then there will be a tendency for computer science curriculi to be overwhelmed by a variety of fragments of detailed, specialized material—and for students of computer science to have negligible opportunity of being exposed to other disciplines, and to areas of intellectual activity where computers do not necessarily play major roles. On the other hand, it is essential for computer science not to grow in isolation, but to actively seek and strengthen intellectual bonds and working contacts with other disciplines. Accordingly, future contributors in the field must develop the ability to communicate and collaborate with people in other areas, and to be sensitive to ideas and concerns outside their discipline. This implies the need for balance between activities within the discipline and outside it. Also, the nature of studies in computer science should reflect the *dynamic character of the field*, and the need to develop in students a capacity for independent learning and growth.

The following set of approaches are suggested in response to the requirements for educational balance and for adaptability to fast changes in the field:

(a) The core of computer science courses that are needed for an undergraduate major should be limited to between a third and a half of the total courses needed for an undergraduate degree. These ratios will be much higher at the graduate level. In any event, students at all levels should be encouraged to take courses in other areas.

(b) An important part of academic planning is to develop (in some detail) options for *combined studies* involving computer science and other disciplines. Undergraduate areas of specialization should be created by combining the computer science core courses with courses in other appropriate disciplines (in particular, with Mathematics, Electrical Engineering, and Management Sciences/Business Administration). This implies a limited core curriculum offered by the computer science unit, and *close curriculum coordination* with the other academic units. Similar curriculum coordination should develop in graduate programs. One of the most effective ways of stimulating contact with other disciplines at the graduate level is to facilitate/induce projects and research which are directly related to advanced computer applications in these disciplines.

(c) An appropriate balance (and coordination) should be established between the lecture courses, the seminars, and the project-oriented courses that are components of a program. Lecture courses should concentrate on fundamentals, on structured presentations of parts of the field, and on general guidance to independent study of the growing body of 'raw data' concerning specific computer systems, languages, and applications. Seminars should focus on specific new developments in the field (exploratory current research, attempts to develop principles and theories) and on discussions of broad issues (e.g., computers and society). Project-oriented courses should provide opportunities for synthesis of information obtained from various sources, for collaboration (with people in the field, as well as with others), and for extensive practical work within the state of the art. An important project course would be an undergraduate, upper-level "*design studio*" where students would work on substantial software designs of operating systems, language processors, interfaces, etc.

(d) The interface between undergraduate and graduate programs in computer science should be flexible—with students moving relatively freely both ways, and courses moving steadily from the graduate to the undergraduate program. This movement of courses does not mean an increase in the number of undergraduate courses, but a restructuring, updating and general strengthening of the undergraduate program—whose overall size should remain relatively stable. One of the reasons for a flexible undergraduate-graduate interface is the fact that most students entering graduate computer science programs still come from different academic disciplines, and their knowledge of computer science varies widely. As undergraduate computer science becomes more widespread, the nature of Masters-level programs will change, and prerequisites for graduate admissions are likely to become more demanding in areas of computer systems and applications. In the meantime, criteria for graduate admissions should remain more flexible on questions of specific subject matter; the emphasis should continue to be on overall undergraduate academic performance (especially in mathematical sciences), personal characteristics such as initiative and inventiveness, and motivation to work in computer science. Because of the dynamic character of the computer field, I expect that we will have for some time a process of curricular changes where the Masters program of today will be essentially included in the Bachelors program of tomorrow. In general, a terminal Masters program in computer science will continue to provide the training needed for advanced professionals in the computer field. Increasingly, Bachelor programs in computer science will also train many of the professionals in the field.

(e) A major part of advanced graduate work in computer science should be devoted to exploratory projects and to *research*. A good research environment is essential. This means an appropriate intellectual climate, good communications, and easy access to computer resources. An important goal, both for faculty and for graduate students, would be to introduce order and cohesiveness in the field, to search for basic principles and general methods, and to design better academic programs in computer science. As part of these activities, new courses should be developed, with the emphasis on fundamentals and also on ways of using computers as integral parts of the preparation and administration of the courses. It would be desirable to have doctoral research develop in the context of such educational projects.

In general, we should recognize that systematization of fundamentals in the computer field, and design of appropriate academic programs is a key responsibility of computer science educators. These activities constitute a process of *continuous self-improvement* of computer science programs. Such a process is an essential part of any academic program; it is of special significance in computer science. Efforts toward academic planning and program improvement should receive support and recognition—the same as direct teaching and other types of research and development. Contributions in these areas will have important implications both for the effectiveness of computer science education, and also for the overall quality and efficiency of work in the computer field.

REFERENCES

1 S AMAREL
Computer science: A conceptual framework for curriculum planning
Communications of the ACM Vol 14 No 6 June 1971

2 *Undergraduate program in computer science*
Department of Computer Science Livingston College Rutgers University New Brunswick N J 08903 April 1971 Available from the Department

3 *Graduate program in computer science*
Department of Computer Science Rutgers University New Brunswick NJ 08903 July 1971 Available from the Department

4 *Descriptions of graduate courses in computer science*
Department of Computer Science Rutgers University New Brunswick NJ 08903 July 1971 Available from the Department

5 S AMAREL
Problem solving and decision making by computer: An overview
in "Cognition: A Multiple View" P L Garvin (ed.) Spartan Books NY 1970

6 W FEURZEIG et al
Programming languages as a conceptual framework for teaching mathematics
Report No 2165 June 30 1971 Bolt Beranek and Newman 50 Moulton St Cambridge Mass 02138

Computer science education—The need for interaction

by M. STUART LYNN

Rice University
Houston, Texas

I would suggest that one of the central problems of computer science today is understanding what it's for. As a consequence, given this hypothesis, from an educational standpoint it is not clear what students are expected to represent when they graduate with degrees in computer science. To the potential employer, the situation becomes even more confused when (in better times) he believes he is recruiting one thing and finds that he has hired something else.

This lack of understanding and eventual misrepresentation is not surprising in a young discipline. Perhaps one of the difficulties faced by computer science is what may be an over-eager attempt to formalize its content and boundaries, and in so doing discouraging the growth, change and vitality which have characterized the last decade. Perhaps, also, there is an increasing role being played by what may be (but should not be) an underlying conflict between the systematic and formalizing demands of science, and the engineering demands by which concepts are made useful to man; this conflict underlies much recent uncertainty in education in other disciplines, and it should not be surprising that it is becoming increasingly present where computing is concerned.

An example, in the Curriculum 68 Report,[1] of where potentially harmful boundaries exist lies in the treatment of numerical analysis. On the one hand, insufficient priority is given to this area, insofar as the proposed course contents are only suggestive of the field and give no insight as to the relationship of numerical analysis to the domain of computer science or to the domain of uses of numerical analysis in computer applications. On the other hand, excessive priority is given to numerical analysis when compared with other uses of mathematical analysis in computer applications: statistics, signal analysis, pattern recognition, control theory, mathematical programming, operations research and others. The proper role of each of these and related areas, and of their interrelationships, needs to be determined by examining the ways in which they are put to use in computer applications, with what priority, and then working backwards to define the appropriate educational structure. It is not clear that the existence of numerical analysis in the computer science curriculum should be an end in itself.

Perhaps too much of computer science has been concerned with formalization. To a degree, such concern is proper; there is clearly a need to develop and understand, systematically, fundamental principles and the interaction of these principles with each other; and a further need to educate in such a way as to encourage growth based upon a formalized body of knowledge. This should not, however, be the end in itself, as appears to be the trend.

This trend, if it exists, is not directly the fault of the Curriculum 68 Report.[1] It occurs partly perhaps because Curriculum 68 is too comprehensive, or too ambitious; and since in practice limited budgets may dictate that only selected parts are implemented, an unbalanced educational structure may be achieved.[2] Perhaps, too, Curriculum 68 is guilty of leaving to chance (see p. 155) what may be the most difficult question of all; how to effect the interaction with other disciplines which is so essential to the vital development of computer science. Amarel[3] has separated computer science activity into its *synthetic* and *analytic* components, the former having a "pragmatic flavor largely responsible for major advances in computers and for great diversity of areas in which computers are being applied", and the latter ". . . providing conceptual guidelines". He advocates a continuous interaction between these two components as essential to a vigorous rate of progress in the field. I would suggest that education in computer science, as structured in Curriculum 68, may be guilty of inhibiting that interaction, not encouraging it.

More positive steps need to be taken to encourage this interaction between the science of computing and the uses of computers. I am not suggesting interaction does not exist, only that it has received decreasing emphasis in recent years. Perhaps such encouragement cannot be achieved solely within the definition of computer science education itself, but requires closer examination of the administrative structures within which computer science operates.

It is likely that the industrial employer will become increasingly selective in its recruiting at all levels. This will be dictated not only by short-term considerations, such as the current state of the economy, but also because he will primarily be looking for graduates who have the breadth to be able to understand and deal with a wide spectrum of his problems, graduates who have the flexibility to modify their areas of interest as needs and priorities change. To the extent that education in computer science prepares students for their future, and for a considerable number this may mean industrial employment, Curriculum 68 needs careful reexamination, since it is not clear that it contributes substantially to this requirement.

A mathematician has been defined as one who, given a problem to solve, solves a whole class of problems of which the original is not a member. It can be argued that there has been a breakdown in the lengthy communication channels between mathematics and the environment in which it operates: whether this is good or bad for mathematics is a matter of point of view. It would, however, be bad for computer science. If students are to be prepared for what may be ahead, they must be given perspective, and a realistic insight into how computer science fits into and relates to the world around it. They should not, to reverse Oscar Wilde's definition of a cynic, be encouraged to know the value of everything and the price of nothing.

REFERENCES

1 *Curriculum 68*
Comm of the ACM 11 1968 p 151

2 R W ELLIOTT
Master's level computer science curricula
Comm of the ACM 11 1968 p 507

3 S AMAREL
Computer Science: a conceptual framework for curriculum planning
Comm of the ACM 14 1971 p 391

Operating systems principles and undergraduate computer science curricula*

by PETER J. DENNING

Princeton, University
Princeton, New Jersey

INTRODUCTION

In the years since 1969, the study of computer systems has assumed a role nearly equal in importance to "theory of computation" and "programming" in computer science curricula. In contrast, computer systems was regarded as recently as 1965 as being inferior in importance to these two more traditional areas of study. This is a significant change in attitude. The harbingers of the change appear in ACM's Curriculum 68,[1] and the speed of its development is demonstrated in the report of Task Force VIII of the COSINE (Computer Science in Electrical Engineering) committee of the Commission on Education of the National Academy of Engineering, entitled "An Undergraduate Course on Operating Systems Principles."[2]

The reasons for this change, the nature of computer-system studies, and the potential impact on future developments will be analyzed in this paper. I shall begin with a brief overview of computer science itself, in order to place the study of operating systems in perspective; I shall then discuss the general need in the computer industry for principles, many of which relate to computer and operating systems; finally, I shall discuss how an operating systems principles course may be organized.

ON COMPUTER SCIENCE

Computer science is the study of the design, analysis, implementation, and application of information-processing procedures.[1,3,4,5,6] In their writings, our educational leaders have consistently stressed the importance of *concepts* and *first principles* in studying the material of computer science. All proposals for computer science curricula—especially undergraduate curricula—have distinguished carefully between courses dealing with first principles and courses dealing with applications. Courses of the former type tend to be considered as "core curriculum" courses, but not courses of the latter type. (It is interesting to note that courses on operating systems were, until very recently, considered as being of the latter type. Even as it has been discovered that operating systems has a set of first principles independent of details depending on the technology, this subject matter has increasingly been considered as worthy core material.)

Courses offered in computer science curricula can be classified (roughly) into four categories: formal systems, programming systems, computer systems, and applications. The distribution of courses among the four areas will depend on the objectives of a given institution. (I believe they should be of equal importance.) *Formal systems* deals with the various mathematical systems for studying computation itself. Some of the topics taught in courses of this category include: the theory of automata, the theory of formal languages, the theory of computability, the theory of complexity of computations, and the theory of numerical computations and error propagation. *Programming systems* deals with the algorithms that perform computations on a practical level. The topics taught in courses of this category include both the concepts of programming languages and the problems of implementing algorithms. Programming language concepts encompass topics like programming language control structures (e.g., sequencing, iteration, conditional transfers, assignments), representing data, use of recursion, use of subroutines, parsing, compilation, and assembly techniques. The study of algorithms includes topics like language-independent algorithm specification, data representation, analysis of algorithms and data structures, proving *a priori* the correctness of algorithms, algorithms for searching and sorting, algorithms for

* Work reported herein was supported in part by NSF Grant GY-6586.

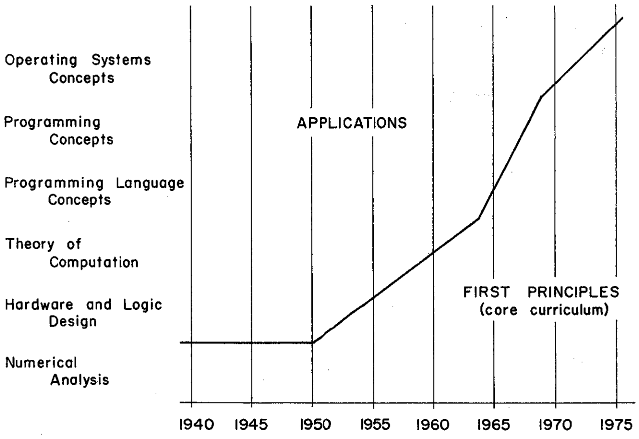


Figure 1—Evolution of first principles

dynamic storage allocation, and heuristic algorithms. *Computer systems* deals with the problems arising when algorithms are implemented on physical devices. Some of the topics taught in courses of this category include design of hardware and logic circuits, organization and design of standard equipment (e.g., processors, memories, peripherals), design of software systems, design of complex systems, control of parallelism and concurrency, control of processor and memory and other resources, analysis of system behavior, modeling the behavior of computing processes. As will be discussed shortly, these three categories deal with what I call the "first principles" of computer science. Topics which are dependent on the present-day technology, or are likely to be of little interest in more than, say, five years, are the subject of courses in the *applications* category; these include discussions of particular languages or particular machines or particular operating systems, systems programming techniques, specialized programming techniques, business data processing, and information retrieval.

The reader will note that Artificial Intelligence has not been mentioned in the above. In many respects, it cuts across all the categories. In many respects, it is an entirely different approach to computer science than the one I have outlined. Many computer science departments offer Artificial Intelligence courses on an elective basis and do not consider them as part of the core curriculum.

When I use the term "operating systems principles," I mean the software, control, modeling, and analysis aspects of the computer-systems part of computer science.

Figure 1 is a very idealized overview of the evolution of the first principles of computer science. Topics are listed along the vertical axis, time along the horizontal; those topics below the curve at a given time are considered as core-curriculum topics at that time. Conceptual approaches to numerical analysis have been with us since the beginning of electronic computing, since these were the primary purposes to which Burks, Eckert, Goldstine, Mauchly, von Neumann and their contemporaries envisioned their machines being applied. Although hardware and combinational logic design concepts using Boolean algebra to describe switching functions had been proposed by Shannon before 1945, this area received a strong impetus from the sequential machine work of Huffman, Mealy, and Moore during the early 1950's. The modern approach to the theory of computation, which emphasizes the relations among abstract machines and abstract languages and computational complexity did not evolve until the mid 1960's and the first texts in this area did not appear until the later 1960's.[7,8] Programming languages were introduced in the later 1950's (FORTRAN in 1956, ALGOL in 1958) but the concepts underlying them were not put down in systematic, textbook form until the mid-1960's.[9] Concepts about algorithms themselves, concepts more or less independent of programming languages, did not become systematized until the late 1960's.[10] And finally, the concepts of computer systems are only recently being put in text form at a graduate level[11] and nothing save a report exists describing them at an undergraduate level.[2] Thus it is evident that the development of a wide range of core material—the first principles of computer science—is quite recent in the era of electronic computing. Since much of the material of major interest to the industry itself—programming and system concepts—has just recently entered the "core area", it is no surprise that the industry has long regarded computer science as being somewhat irrelevant. I do think we can expect a change in this attitude.

On the basis of Figure 1, one can see why the majority of computer science departments were not established until after 1965. Academia has always been reluctant to establish degree programs in subject areas with no discernible broad base of first principles.

ON THE NEED FOR PRINCIPLES TO BE APPLIED IN THE COMPUTER INDUSTRY

As mentioned, Figure 1 illustrates why the industry has tended to hold computer science education in disregard, i.e., because the topics of interest to it (concepts of programming and design of systems) have not been treated on a sound conceptual basis in curricula. Looking at Figure 1 from a different perspective turns up another interesting observation: For some twenty years, the leaders of industry (management) have had to proceed without a set of first principles in the areas of major interest to them. This perhaps explains (in part) why so much difficulty has been experienced in getting "third generation" computer systems operational: There had been no systematic set of concepts according to which designers and programmers could be guided. This had left the industry in the rather uncomfortable situation of being accused in the public media of misusing, or condoning the misuse of, computers. Witness, for example, the great public outcry with respect to privacy and protection of information.

In many respects, then, the allegations that the computing industry is "adrift" and "unprincipled" have some basis in fact. The allegations that computer science education has not been of much help to the industry likewise have some basis in fact. We cannot, however, afford to wait ten or twenty years for the present crop of computer science students to move into the leadership positions of industry. Those presently in the leadership positions must not only be made aware of the existence of principles, but they must be made aware of the principles themselves.

In a recent statement,[12] ACM President Walter Carlson recognized this problem, saying "... There is too little systematic codification of fundamentals (i.e., first principles). There is almost no communication between the research people and system designers or operators. There is a widening sense of frustration among academics over misuse (or lack of use) of proven computer technology in practical operations." Figure 1 demonstrates that at least a basis exists for reversing the trend addressed by Carlson's first point.

To illustrate Carlson's third point, let me discuss three examples of common misconceptions attributable to a lack of understanding of the first principles of computer science: (1) The "software problem," i.e., the increasingly high costs of software development and lack of quality control over software packages, is a long way from being solved. (2) The "protection problem," i.e., that of guaranteeing the privacy, security, and integrity of information, is yet to be solved adequately. (3) The "computer utility" was a fad which, thankfully, is dead. It would astonish some managers to know the facts: Solutions to the software and protection problem exist now—and have existed, in computer utility research projects, since at least 1965! (I will elaborate below.) While much work remains to be done, I think it is safe to say that these solutions are adequate for present-day purposes. The solutions to these problems evolved in the design and use of such systems as the CTSS (Compatible Time Sharing System) and its successor MULTICS (Multiplexed Information and Computing Service) at MIT. Both these systems are regarded as first steps in the evolution of the computer utility, and rely on the computer-utility environment for their viability; thus, the attitude that the computer utility was a fad is in fact an attitude hostile to the solutions of the two problems. That so many who hold leadership positions in the computer industry have failed to recognize that CTSS and MULTICS implement solutions to these problems—even as undergraduate students exposed to an operating systems principles course seem to have no such difficulty—further reinforces my conviction that the first principles of computer science are not widely disseminated or appreciated.

The solution to the software problem in CTSS and MULTICS is based on the concepts of *programming modularity* and *sharing of information;* the requisite mechanism took the forms of the CTSS file system and the MULTICS virtual memory.* The solution to the protection problem in MULTICS is based on the concepts *protection rings* (or "domains") and on *controlled access to information.*** An important aspect of these two solutions is, neither can be accommodated on most systems presently on the market without major alterations in hardware and software architecture. It is the failure of management to understand the principles on which the solutions are based that has made them unable to recognize that CTSS and MULTICS solve these problems, and has led them to market systems in which technically sound solutions to the software and protection problems are not possible.

The foregoing criticisms of management are meant to be constructive in the sense that managers (and other elements of the computing profession as well) cannot play their roles to perfection without a thorough understanding of the principles of computer science. And, because it is only recently that the first principles have become systematized, it is difficult to expect managers to interpret new developments in the light of guiding principles.

ON THE IMPORTANCE OF COMPUTER SYSTEMS CONCEPTS

I have devoted a considerable amount of space to emphasizing the first principles of computer science and

* A description of the CTSS file system can be found in Wilkes' remarkably concise 96-page book,[13] a description of the MULTICS virtual memory can be found in the paper by Bensoussan, Clingen, and Daley.[14] Both these descriptions make it adequately clear that these systems, in their own contexts, have solved the software problem.

** The concepts of protection systems are discussed by Lampson,[15] and by Graham and Denning;[16] systems implications of protection are discussed by Wilkes[13] and by Schroeder and Saltzer.[17]

analyzing the effects in the computer industry of having no first principles. I have done this to emphasize how the first principles of computer systems have a natural place in computer science. Aside from these larger considerations, there are several reasons why a systematic presentation of the first principles of computer systems is important in and of itself.

First, the so-called case-study approach to teaching computer systems has been more or less unsuccessful wherever it has been tried: The student becomes so immersed in the particulars of the given system that he cannot distinguish important principle from irrelevant detail. In my limited experience, I have met only with success using the "modeling and analysis" approach. Thus, computer system principles are useful in the practical task of teaching others about computer systems.

Second, it is increasingly evident throughout the computer industry that we literally can no longer afford to develop ever larger, ever more complex systems without solid notions of how they are to be structured or how they are to behave. We can no longer afford to put together systems which are extremely wasteful of their own resources. We can no longer afford *not* to apply the concepts and principles in practice.

Third, for some inexplicable reason, the design of complex software and hardware systems has been considered traditionally as an "application" problem, a "technology-dependent" problem. In contrast, it is now being realized that designing complex systems—systems which operate as intended, whose correctness and behavior can be established *a priori*, whose performance can be monitored easily—is an intellectual task worthy of our best minds. (In other words, the computer-systems area has inherited one of the most important problem areas of computer science.) This realization has stimulated increasing amounts of research in computer system modeling and analysis, a principal result of which has been the emergence of a teachable, comprehendable body of operating systems principles. Much of "computer system theory" is concerned in one way or another both with managing complexity in software and hardware systems and with complex algorithms involving parallel processes, so that it is relevant to many practical problems now facing the industry.

Fourth, and perhaps of the most long-term significance, there is an ever widening appreciation of the view that our world has become a vast real-time system whose complexity is beyond the reach of the unaided human mind to understand. It is not difficult to identify the essential elements of information systems in business systems, economic systems, urban systems, and even social systems. Again and again we see decisions being taken to regulate such systems as these which, despite the best of intentions, often turn out to have the opposite of the intended effect. Jay Forrester has called this phenomenon the "counterintuitive behavior" of complex systems.[18] It is explained by the inability of the unaided mind fully to grasp why the long-term effect of a policy is often the opposite of its short-term effect, or fully to comprehend the complexities of, and interrelations among, the various parameters that influence a system. Forrester's simulation experiments of urban and business systems show that the intended result may normally be obtained only when *all* the controllable parameters of a system are governed by a single policy, not when each such parameter is controlled individually. This type of phenomenon is hardly new in the experience of computer system designers. As computer system "theorists" axiomatize information systems, developing systematic approaches both for managing complexity and for guaranteeing *a priori* that a system will operate as intended, the results of their efforts should be applicable to the solutions of problems in social, urban and other noncomputer information systems.

ON THE APPROACH

As I have stated, I call this the modeling-and-analysis approach to studying operating systems, to distinguish it from the case-study approach. The material can be organized and presented as a sequence of abstractions together with examples of their applications, each abstraction being a principle from which most implementations can be deduced. The concept-areas (in which teachable abstractions exist) are:

procedure implementation
concurrent processes
memory management
name management
resource allocation
protection

Experience shows that the most coherent treatment is obtained if the topics are organized according to this list of concept-areas.

The COSINE report[2] goes into some detail instructing instructors how to present these abstractions; a COMPUTING SURVEYS paper[19] explores them in some detail from a student's viewpoint, the Coffman-Denning text[11] treats the mathematical analysis related to them. Accordingly, I shall restrict myself here to outlining their content.

The background required of students undertaking this type of course should include: a working knowledge of programming language features; an elementary understanding of compilation and loading processes; processor organization and operation, including interrupts; memory organization and operation; and data structures such as stacks, queues, arrays, and hash tables. This background could be obtained from a course on programming languages (e.g., ACM's Course I-2)[1] and a course on computer organization (e.g., ACM's Course I-3).[1] A data structures course (e.g., ACM's Course I-1)[1] is helpful but not necessary.

The course itself is related to ACM's systems programming course (ACM Course I-4)[1] but differs in at least two significant ways. First, the ACM outline suggests a "descriptive," case-study approach whereas this course is organized along conceptual lines. Second, ACM's course emphasizes techniques of systems programming whereas this course emphasizes the principles of system organization and operation. This shift in emphasis is made possible by new developments, since the ACM report predates the appearance of much of the modeling and analysis material on which this course is based.

The instructor of this course will find it useful to introduce the subject by pointing out how, despite the wide range of operating systems types and capabilities, there is a set of characteristics common to these systems. These characteristics include: (1) *concurrency*, i.e., many activities proceeding simultaneously, (2) *sharing of resources* and the existence of a centralized resource allocation mechanism, (3) *sharing of information*, (4) *protection* for information, (5) *long-term storage* of information, (6) *multiplexing*, i.e., the technique of switching a resource (rapidly) among many requestors so that it is assigned to at most one at a time, (7) *remote conversational access*, (8) *nondeterminacy*, i.e., unpredictability of the order in which events will occur, and (9) *modularity* in design and operation of systems. It is important to point out that, for pedagogic reasons, the course material is presented (more compactly) according to the six concept-areas stated earlier, and not directly among the lines of these nine common characteristics.

The area of procedure implementation is important because the reason computer systems exist at all is to provide an efficient environment for executing programs. The notion of procedure is important because of its close relation to programming modularity. The basic concepts—pure procedure, procedure activations and activation records, parameters, local and nonlocal references—should be presented first. Then the main concept, "procedure in execution" (i.e., a procedure which has been called but has not yet returned) and its implementations, is presented. An abstract description of "procedure in execution" is a pair of pointers (i, r) where i is an instruction pointer and r an activation-record pointer (local environment pointer). Every implementation must solve three problems: (1) allocating and freeing storage on procedure call and return, (2) interpreting local references, i.e., those to objects in the activation record, and (3) interpreting nonlocal references, i.e., those to objects in other activation records. The implementations of "procedure in execution" in languages like FORTRAN, ALGOL, or PL/I can be deduced from these concepts by considering the restrictions imposed by these languages.

The area of concurrent (parallel) processes is important because one of the purposes of an operating system is controlling many, independently-timed activities. A "process" can be regarded as a "program in execution" and has an abstract description much like "procedure in execution." "Parallel Processes" is the notion that, at any given time, more than one program will be observed to be somewhere between its initiation and termination points. There are four process control problems of principal concern: (1) *determinacy*, i.e., the property that the result of a computation by cooperating processes on common memory cells is independent of their relative speeds, (2) *freedom from deadlock*, i.e., the property that allocation of resources is controlled so that at no time does there exist a set of processes, each holding some resources and requesting additional resources, such that no process's request can be satisfied from the available resources, (3) *mutual exclusion*, i.e., the property that, of a given set of processes, at most one is executing a given set of instructions at any time, and (4) *synchronization*, i.e., the property that a process, whose progress past a given point depends on receiving a signal from another, is stopped at that point until the signal arrives.

The area of memory management is important because every practical computer system incorporates a hierarchy of storage media characterized by various compromises among access time, capacity, and cost; a set of policies and mechanisms is required to increase system efficiency by arranging that the most frequently accessed information resides in fast access memory. The abstractions used here are those of "virtual memory": address space, memory space, and address map. The common implementations of virtual memory (e.g., paging, segmentation) can be deduced from these abstractions by considering factors such as efficiency of mapping operations and efficiency of storage utilization. Once the implementations have been studied, one can study the policies for managing them. Finally, it is straightforward to generalize the discussion to the implementations and policies of multiprogramming and of auxiliary memory problems. The foregoing development will lead to a computational storage system (i.e., that part of the memory system in which references are interpreted by the hardware) which presents to each programmer a large, linear address space.

The area of name management is important because a linear address space such as provided by the preceding development has inherent limitations from the standpoint of programmers and system users. It cannot handle growing or shrinking objects, provide different contexts in which processes may operate, allow for sharing or protecting objects, or implement a long-term storage system in which objects may reside independently of any context. In other words, linear address space cannot support modular programming to the extent required by today's system objectives. These limitations can be overcome by extending the memory system to allow programmers to define objects of variable size, assign names and (variable) contexts to these objects, allow shared access to objects, and specify dynamically which subset of a universe of objects should participate in a computation. That part of the memory system which implements these new objectives is called the "long-term storage system"; it may be distinct from the computational storage system or it may be one and the same. Computers implementing a virtual memory and a file system are examples of the former, computers implementing segmentation are examples of the latter. In either case, a global (system-wide) scheme for naming objects must be devised (in order to allow sharing), the (directory) tree hierarchy with "pathnames" being most common (in this case the long-term storage system provides a tree-structured space of objects in addition to the linear space or spaces provided by the computational storage system).

The area of resource allocation is important partly because the complexities of the interactions among all the processes in the system dictate that a central policy regulate resource usage to optimize performance, and partly because one effect of implementing the previous abstractions is to hide machine details from users, placing the burden of allocation decisions squarely on the system. One can distinguish long-term from short-term policies, the latter being of primary concern here. One can model a process (from a resource-allocation view) as cycling among the "demand-states" ready, running, blocked; correspondingly one finds a network of queues with feedback in the computer system. Processes are distributed among the queues according to demand-states, and change queues according to changes in demand-state. In terms of this, one can study specific queueing policies and overall network control policies; one can study the meaning of concepts like "system balance" and "thrashing"; and one can study what evaluation techniques are applicable. Models of program behavior and the use of statistical analysis are important to an understanding of resource allocation and are used throughout.

The area of protection is important in any system where there may reside procedure and data belonging to more than one individual. It must not be possible for one process to disrupt or corrupt service to others, and access to information (especially if confidential or proprietary) must be permitted only under appropriate authorization. The abstract model of a protection system includes: (1) a set of "domains" or "protection contexts," i.e., sets of constraints according to which a process may access objects, (2) a set of "objects," i.e., everything to which access must be protected or controlled (including the domains), and (3) a mechanism that both specifies and enforces access rules by processes to objects, the type of access depending on the domain with which the process is associated. The mechanism can be specified in the form of an "access matrix" (e.g., if A is the access matrix, $A[d, x]$ specifies the types of access a process associated with domain d has to object x). Most of the implementations of protection found in practice can be deduced from this model.

There remain certain issues that have not been treated definitively in the literature but which nonetheless are of central importance in computer system operation and design. These include: reliability, performance evaluation, design methodologies, and implementation strategies. They must, unfortunately, be relegated to a pedagogically inferior position at the end of the course. I should emphasize that this reflects the absence of teachable material, not the importance of the issues. As viable abstractions in these areas are evolved, they will be welcome additions to the course.

CONCLUSIONS

Operating systems principles can be regarded as the study of complex algorithms comprising parallel activities. This paper has reviewed and analyzed the importance of a significant change in attitude: The assumption of computer operating systems principles into the core of computer science. The analysis of this change was done in the light of the evolution of the "first principles" of computer science, of the need for these principles to be applied in the computer industry, and of the increasing need for systematic ways of dealing with complex, real-time information systems. An outline for a course on operating systems principles was described, the concepts being chosen for inclusion in the course on the basis both of demonstrated utility in practice and of their being straightforward generalizations of widely accepted viewpoints.

The first twenty-five years of the computer industry, years of remarkable achievement, have not been without their problems. Now that computer science education is maturing, we should be able to expect closer cooperation between universities and industry in solving these problems.

REFERENCES

1 ACM Curriculum Committee on Computer Science (C^{3}S)
Curriculum 68: Recommendations for academic programs in computer science
Comm ACM 11 3 March 1968 151-197

2 COSINE Task Force VIII
An undergraduate course on operating systems principles
(The members of the task force were: Peter J. Denning, Jack B. Dennis, A. Nico Habermann, Butler W. Lampson, Richard R. Muntz, and Dennis Tsichritzis.) June 1971. Available free of charge from: Commission on Education National Academy of Engineering 2101 Constitution Avenue NW Washington DC 20418

3 S AMAREL
Computer science: a conceptual framework for curriculum planning
Comm ACM 14 6 June 1971 391-401

4 G E FORSYTHE
A university's educational program in computer science
Comm ACM 10 1 Jan 1967 3-11

5 R W HAMMING
One man's view of computer science
J ACM 10 1 Jan 1969 3-12

6 A L PERLIS
University education in computing science
Proc Stony Brook Conf Academic Press 1968 70ff

7 J E HOPCROFT J D ULLMAN
Formal languages and their relation to automata
Addison-Wesley 1969

8 M M MINSKY
Computation: Finite and infinite machines
Prentice-Hall 1967

9 I FLORES
Computer programming
Prentice-Hall 1966

10 D E KNUTH
The art of computer programming
Addison-Wesley Vol I 1968 Vol II 1969

11 E G COFFMAN JR P J DENNING
Operating systems theory
Prentice-Hall to appear

12 W M CARLSON
President's letter: reflections on Ljubljana
Comm ACM 14 10 Oct 1971

13 M V WILKES
Time sharing computer systems
American Elsevier 1968

14 A BENSOUSSAN C T CLINGEN R C DALEY
The MULTICS virtual memory
Proc 2nd ACM Symposium on Operating Systems Principles Oct 1969 30-42

15 B W LAMPSON
Protection
Proc 5th Annual Princeton Conference on Information Science and Systems Department of Electrical Engineering Princeton University Princeton New Jersey 08540 March 1971

16 G S GRAHAM P J DENNING
Protection: principles and practice
Proc AFIPS Conf 40 Spring Joint Computer Conference 1972

17 M D SCHROEDER J H SALTZER
A hardware architecture for implementing protection rings
Comm ACM 15 3 March 1972

18 J W FORRESTER
Urban dynamics
MIT Press 1969

19 P J DENNING
Third generation computer systems
Computing Surveys 3 4 Dec 1971

Theory of computing in computer science education

by PATRICK C. FISCHER

University of Waterloo
Waterloo, Ontario, Canada

INTRODUCTION

Theory of computing means the abstract study of the nature of computation and computing devices. By convention, the terminology is usually applied in a narrower sense to exclude numerical analysis. Thus, theory of computing includes the theory of finite automata, formal languages, computability, computational complexity, and some aspects of switching circuit theory and logical design. The deviation from the literal meaning of the term may have occurred because numerical analysis was already a well-established subject when the other components of this area were in their infancy. On the other hand, it may be a reflection of the emphasis on discrete mathematics by workers in theory of computing, in contrast to the preponderance of continuous methods in numerical analysis.

In *Curriculum 68*, theory of computing was represented by the following courses.*

B3	Discrete Structures
I6	Switching Theory
I7	Sequential Machines (Automata Theory)
A1	Formal Languages and Syntactic Analysis
A7	Theory of Computability

CURRICULUM TRENDS

Lower-level courses

The courses B3, I6 and I7 above have been remarkably stable, perhaps because of the relative distance of their content from the frontiers of research. Perhaps some minor changes might be considered: Course B3 could include some material on the first-order predicate calculus rather than just the propositional calculus, and Course I7 still lacks a textbook which is teachable and has a good balance between intuition and rigor. Nevertheless, the 1968 descriptions for these courses are still a very good approximation to current course offerings in a number of university and college departments of computer science.

Formal languages

Turning now to the advanced-level courses, one finds more updating needed. Developments in formal language theory have taken place in both the area of parsing methods and the study of abstract families of languages. An updated course A1 would probably do well to assimilate a greater proportion of the new material in the former area, thus shifting the overall orientation of the course more toward the practical side. In order to make room for this recent material, a few lectures worth of the most basic material, e.g., the notion of a formal grammar as a generator of a language and some of the equivalent interpretations in terms of abstract machines, could be introduced earlier as part of the course I-X proposed below.

Theory of computability

A considerable amount of activity has taken place within the area covered by Course A7, and much of the new material is already entering the computer science course offerings of several universities. The new material lies principally in three areas:

1. computational complexity,
2. analysis of algorithms,
3. program schemata.

We will discuss these areas briefly in order to define their scopes, but will not attempt a survey of the many interesting results which have been obtained.

* *Comm. ACM 11*, 3 (March, 1968), pp. 151–197.

Computational complexity

The first of these areas dates back to 1960 when M. Rabin suggested that one could study the classification of computable functions according to the degree of difficulty of computing them.^{B28} (Until then, the main efforts in computability theory were, in fact, directed toward understanding and classifying the uncomputable functions in order to gain insight into the foundations of mathematics.) Thus, this area is the oldest of the three and was the first to split off from the traditional computability theory (often called recursive function theory), which began in the 1930's.

In computational complexity studies, the most common overall approach is to consider programs which are "efficient" in the sense that they either do not exceed a bound on resources available at execution time or do not exceed a bound on those available at compile time. Thus, one either considers all programs which are sufficiently "fast" or sufficiently "small" and determines which functions can be computed by such programs. The classes of functions obtained relative to a given resource bound are called "complexity classes," and one principal aim of this area is to study their mathematical properties. Examples of resources related to execution of a program which have been studied are: number of steps on a Turing machine, memory cycles on a random-access machine, total memory space needed on a typical computer, number of bitwise operations on any computer. Examples of resources related to compilation of a program are: program size (in bits), depth of nesting of DO loops in a FORTRAN-like language, number of blocks in a program.

In addition, the features of all computations with resource bounds have been axiomatized by M. Blum, and a number of machine-independent results have been proved. Thus, mathematical rigor has reinforced the programmer's intuition in verifying that for any computable function there are arbitrarily bad (e.g., slow or huge) programs which do, in fact, compute the function. On the other hand, the "best" programs for a function cannot always be effectively found. Furthermore, some functions simply have no best programs.

The notion of "time-space trade off" is beginning to be the object of study via the consideration of complexity of finite functions (i.e., functions which need to be evaluated for only a finite number of inputs). Until recently, finite functions had been dismissed as trivial, since all finite functions can be computed quickly via programs with large tables built into them. By taking program size into account, a rich enough structure is available to permit the investigations of this subarea. Another related concept is that of randomness, since a sequence can be considered to be essentially random if a program to print it involves the equivalent of storing the entire sequence in the program. Conversely, a non-random sequence is one which can be predicted and therefore computed by a relatively short program.

The techniques used to prove theorems in computational complexity theory tend to be closely related to techniques used in automata theory and recursive function theory. Thus, proofs may involve diagonal constructions, counting arguments or analysis of rate-of-growth of resource needs. The results in this area have had relatively little direct practical application in the sense that the algorithms produced generally compute functions of only theoretical interest. However, a great deal of understanding about the nature of computation and the relationship between machine organization and computational efficiency is being obtained.

Analysis of algorithms

This area has evolved as a branch of computational complexity but has recently acquired its own distinctive character as well as a sizable group of converts. (It is still new enough to have no standard name; other names include "efficient algorithms," "optimal algorithms," and "concrete computational complexity.") Again, interest is in the degree of difficulty of computing functions. However, one now tends to focus on a given function (or a relatively small class of functions) and to attempt to determine the complexity of the best algorithms for computing it, if such exist and can be found. If optimal algorithms are not known, lower bounds on complexity measures are investigated and near-optimal algorithms sought. Examples of functions or procedures considered include: sorting, determination of the median of a sample, polynomial evaluation, matrix multiplication, checking graph isomorphism.

In analysis of algorithms, the measures of complexity are all oriented toward execution efficiency rather than program structure. Examples of measures commonly used are: the number of multiplications and/or divisions executed, number of all arithmetic operations, number of data references, number of data comparisons. The measures thus involve quantities which are basic in most programming systems. Furthermore, by considering only programs which compute a given function, rather than studying all programs possessing a certain degree of efficiency (regardless of what they compute efficiently), one limits himself to programs computing functions of practical interest, and his search for an algorithm which is both efficient and useful follows more closely that of the programmer than the theoretician. For these reasons, analysis of algorithms has a considerably more practical flavor than (other) computational complexity theory.

(Some of this aura of practicality is, of course illusory. Minimizing the number of multiplications may not minimize cost, when eliminating one multiplication adds 10 addition operations. Furthermore, simply counting the number of arithmetic operations tacitly assumes, for example, that addition of very long numbers costs no more than addition of short numbers. Such assumptions are just as idealistic as assuming that minimizing steps on a multitape Turing machine will minimize cost on a real computer, but they have more popular appeal since they do not limit one to hardware with a very low market penetration.)

The methods used in analysis of algorithms tend at times to be *ad hoc* in nature, especially when searching for new and better algorithms. The techniques for establishing lower bounds tend to come from combinatorics and numerical analysis, and a background in these areas appears to be more important for the algorithm analyst than a background in other theory of computing.

Program schemata

Research in program schemata is also on the upswing although the total activity in this area is probably not as high as in the other two. The main objects of study are programs in which the particular domain of data and the meanings of the primitive operations on that data are not specified. One example of a program schema would be a flow chart in which the contents of the various computation and decision boxes are left unspecified. One then studies the relationships between the "uninterpreted schema" (e.g., the flow chart) and the programs resulting when various functions are "plugged in" to the uninterpreted portions of the schema (e.g., the flow chart boxes).

In its purest form (most features of the model uninterpreted), the study of program schemata is closely related to work in language definition and program validation. As one builds more structure into the model, allowing some features to have fixed interpretations (e.g., a box which computes $\lambda x[2x]$) and others to be uninterpreted, this area tends to behave more like a branch of computability theory. However, instead of computable functions, one now considers computable functionals, i.e., mappings from functions to functions rather than from numbers to numbers. For example, a flow chart with two boxes in which the output of the first box becomes the argument for the second might be viewed as the functional producing composition of two functions when the functions are given as inputs. Thus, if the first box were to be interpreted as computing a function g and a second box as computing f, then the whole program would compute $\lambda x[f(g(x))]$.

The techniques for proving theorems about program schemata tend to be similar to those of classical computability theory, and the models used have their roots in common programming languages.

SUGGESTED CURRICULUM MODIFICATIONS

Basic computability theory

A few minor curriculum changes were suggested at the beginning of the previous section. However, for the course offerings in computability theory, the modifications should be more extensive. It is proposed that the more basic and machine-related material in the present Course A7 be combined with a short introduction to formal language theory and presented as a new intermediate level Course I-X. The choice of material to put into I-X was based on an assessment of relevance to computer science, estimated present development and future stability, and availability of textbook material. By teaching the material in Course I-X at the undergraduate (or first-year graduate) level, one will have more flexibility with respect to the advanced-level Course A1 on formal languages and in new courses in the previously described areas of computability theory. Fairly detailed suggestions are made below for Course I-X, following the format of *Curriculum 68*.

The remainder of the material in Course A7 included both recursive function theory and computational complexity material. However, much of the new material in analysis of algorithms and program schemata is more relevant to computer science than that in recursive function theory. To be sure, a student would still benefit from the more traditional approach since understanding the historical origins of a subject helps place the material in a proper perspective. Furthermore, the techniques traditionally used are applicable to problems in computational complexity theory and often in theory of program schemata. However, there is now much more than enough material in computational complexity, analysis of algorithms and program schemata to fill a term course, thus creating a pressure to displace the less relevant material. Therefore, the recursive function theory ought to leave the computer science curriculum recommendations. (It will still be taught in mathematics departments, of course.) It is encouraging that so much good work has been done on problems which have their roots in computer science in so short a period of time.

Course I-X: Introduction to computability theory and formal languages

Approach

This course is designed to introduce the advanced undergraduate or first-year graduate to formal systems of computation (including formal languages) and to give him a basic understanding of the nature of the unsolvability phenomena which pervade attempts to deduce correctness of programs, equivalence of algorithms, etc.

Prerequisite: Course B3 or a course in logic or algebra.

Content

1. The intuitive notion of effective computability as involving the existence of an algorithm. The Turing machine as one precise formal model for the notion. Equivalence of the computing power of the standard class of Turing machines with classes obtained by varying the control structure of the machine (e.g., Wang's single-address serial memory machines) and the data structure of the machine (e.g., more than one tape, multiheaded tapes, n-dimensional tapes, nonerasing tapes, random-access storage, etc.).

2. Universal Turing machines, i.e., the construction of an interpretive program to simulate other Turing machine programs.

3. The unsolvability of the halting problem; the effective production of counterexamples for alleged halting problem solvers. Some other unsolvable problems (blank tape halting problem, printing problem, etc.). Some solvable problems (computations on bounded space, etc.).

4. Numerical approaches to computability. Primitive recursive and recursive functions. Encodings of lists of integers into single integers (pairing functions) and encodings of computations (arithmetization).

5. Other models for digital computation: program machines, loop machines, machines with pushdown stores, stacks, counters, etc. Simulation of Turing machines by machines with only two counters.

6. Post combinatorial systems and formal grammars as a special case. Production notation for specifying grammars. Recursively enumerable, context-sensitive, context-free and finite-state grammars. Examples of languages of varying complexity, e.g., a^nb^n, $a^nb^nc^n$. Proof that $a^nb^nc^n$ is not context-free.

7. Grammars vs. abstract machines. Representations of the above in terms of Turing machines, linear bounded automata, one-pushdown machines, finite-state automata, respectively.

8. Unsolvability of the Post correspondence problem. Applications to unsolvability problems in formal languages.

References—Course I-X

The richest sources of material for this course are Minsky and Hopcroft-Ullman.[A8,A6] The former has good, readable coverage of 1-5, above, and part of 6. The latter has a more formal approach and covers 6-8 very well.

A1 S AANDERAA P C FISCHER
The solvability of the halting problem for 2-state Post machines
J ACM 14 1967 pp 677-682

A2 M DAVIS
Computability and unsolvability
McGraw-Hill New York 1958 210 pp

A3 P C FISCHER
On formalisms for Turing machines
J ACM 12 1965 pp 570-580

A4 S GINSBURG
Mathematical theory of context-free languages
McGraw-Hill New York 1966 243 pp

A5 H HERMES
Enumerability, decidability, computability
Academic Press New York 1965 245 pp

A6 J E HOPCROFT J D ULLMAN
Formal languages and their relation to automata
Addison-Wesley Pub Co Inc 1969 242 pp

A7 S C KLEENE
Mathematical logic
Wiley New York 1967 398 pp

A8 M L MINSKY
Computation: Finite and infinite machines
Prentice-Hall Englewood Cliffs NJ 1967 317 pp

A9 J MYHILL
Linear bounded automata
WADD Tech Note 60-165 Wright-Patterson Air Force Base Ohio 1960

A10 E L POST
Recursive unsolvability of a problem of Thue
J Symbol Logic 11 1947 pp 1-11

A11 H ROGERS JR
Theory of recursive functions and effective computability
McGraw-Hill New York 1967 482 pp

A12 J C SHEPHERDSON H E STURGIS
Computability of recursive functions
J ACM 10 1963 pp 217-255

A13 B A TRAKHTENBROT
Algorithms and automatic computing machines
Translation by J Kristian J D McCawley and S A Schmitt
D C Heath Boston 1963 101 pp

A14 A M TURING
On computable numbers, with an application to the Entscheidungsproblem
Proc London Math Soc Ser 2 42 1936-1937 pp 230-265
Corr ibid 43 1937 pp 544-546

A15 H WANG
A variant to Turing's theory of computing machines
J ACM 4 1957 pp 61-92

Advanced computability theory

No single solution is proposed for taking Course A7, now bereft of both basic computability theory and recursive function theory, and refurbishing it with more relevant material while retaining the computational complexity theory previously contained there. One reasonable approach would be to convert the course into a survey-in-depth of all three areas described above. The most generous answer (for devotees of computability theory) would be to convert A7 into a course on computational complexity theory and to establish new courses in both algorithm analysis and program schemata. This, in fact, is the current situation in the graduate program at Waterloo. However, graduate course offerings are obviously highly dependent upon the interests and quantity of faculty members. (Until the current academic year Waterloo offered courses equivalent to I-X and a single graduate term course in computability theory. The latter course took the survey-in-depth approach.)

Activity in analysis of algorithms has grown so rapidly that the accretion of new results and interesting algorithms has far exceeded progress in determining underlying principles. Thus, it will be difficult to keep a course in this area from becoming a collection of programming tricks. Nevertheless, courses in analysis of algorithms have already sprung up at a number of universities and are attracting respectable enrollment figures. Furthermore, the potential value of such a course for persons who think they like programming and dislike all theory outweighs the difficulties due to the roughness of the area and lack of adequate pedagogical material. Thus, it is quite likely that many schools will choose to offer a course in analysis of algorithms regardless of whether or not the computational complexity and program schemata theory are covered as well. If this turns out to be the case, the best solution for Course A7 would be to embrace both of these latter areas.

A list of topics and references in each of the three areas is given below. In view of the discussion just concluded, these are not to be interpreted as specific recommendations for courses but only as a sample of material in each of the areas.

ADVANCED TOPICS AND REFERENCES

As stated previously, the topics and references presented below are not all inclusive and are *not* suggestions for three separate courses.

Computational complexity

Topics

1. Complexity classes defined in terms of Turing machine time bounds, space bounds, and tape reversals.[B14,B18,B21,B34]

2. Models for real-time computation, Turing machines with many tapes vs. one or two tapes, Turing machines with many heads per tape vs. one head per tape.[B13,B24,B29,B32]

3. Random-access register machines, iterative arrays, n-counter machines and real-time countability, other computing devices.[B2,B9,B15,B16,B36]

4. Axiomatic approaches to machine structure, bounded action and bounded activity machines.[B10,B35]

5. Complexity classification by functional definition, primitive recursive functions, the Grzegorczyk classes, honest functions, loop program machines.[B8,B26,B30,B31]

6. Axioms for machine-dependent complexity theory, existence of arbitrarily bad programs and arbitrarily hard-to-compute functions, speed-up theorem.[B3]

7. Machine-independent complexity classes, uniform hierarchy (compression) theorem, honesty theorem, gap theorem, union theorem.[B20]

8. Complexity of finite functions, expressive power of programs, random sequences.[B4,B7,B25]

References—Computational complexity

In general, only material appearing in standard journals and written in English is given. Thus, a number of recent results and papers in foreign languages, principally Russian, containing relevant material have been omitted. Fairly comprehensive coverage of machine-independent complexity theory as well as some additional references are found in the paper by Hartmanis and Hopcroft.[B20] Two previous references are also relevant to this area.[A9,A12]

An extensive "Bibliography on computational complexity" was compiled by M. I. Irland and the author in October, 1970, but has not been published. A limited number of copies are available as Research Report CSRR 2028, Department of Computer Science, University of Waterloo.

B1 A V AHO J E HOPCROFT J D ULLMAN
Time and tape complexity of pushdown automaton languages
Information and Control 13 pp 186-206

B2 A J ATRUBIN
A one-dimensional real-time iterative multiplier
J ACM 14 1965 pp 394-399

B3 M BLUM
A machine-independent theory of the complexity of recursive functions
J ACM 14 1967 pp 322-337

B4 M BLUM
On the size of machines
Information and Control 11 1967 pp 257-65

B5 M BLUM
On effective procedures for speeding up algorithms
J ACM 18 1971 pp 290-305

B6 R V BOOK S A GREIBACH
Quasi-realtime languages
Math Systems Theory 4 1970 pp 97-111

B7 G J CHAITIN
On the length of programs for computing finite binary sequences
J ACM 13 1966 pp 547-569

B8 A COBHAM
The intrinsic computational difficulty of functions
Proc 1964 Int'l Cong on Logic Methodology and Philosophy of Science North Holland Amsterdam 1965 pp 24 30

B9 S N COLE
Deterministic pushdown store machines and real-time computation
J ACM 18 1971 pp 306-328

B10 S A COOK S O AANDERAA
On the minimum computation time of functions
Trans Amer Math Soc 142 1969 pp 291-314

B11 M J FISCHER A L ROSENBERG
Real-time solution of the origin-crossing problem
Math Systems Theory 2 1968 pp 257-63

B12 P C FISCHER
Generation of primes by a one-dimensional real-time iterative array
J ACM 12 1965 pp 388-394

B13 P C FISCHER
Turing machines with a schedule to keep
Information and Control 11 1967 pp 138-46

B14 P C FISCHER
The reduction of tape reversals for off-line one-tape Turing machines
J Computer Syst Sci 2 1968 pp 136-47

B15 P C FISCHER A R MEYER A L ROSENBERG
Counter machines and counter languages
Math Systems Theory 2 1968 pp 256-83

B16 P C FISCHER A R MEYER A L ROSENBERG
Time restricted sequence generation
J Computer Syst Sci 4 1970 pp 50-73

B17 J HARTMANIS
Computational complexity of one-tape Turing machine computations
J ACM 15 1968 pp 325-39

B18 J HARTMANIS
Tape reversal bounded Turing machine computations
J Computer Syst Sci 2 1968 pp 117-35

B19 J HARTMANIS
A note on one-way and two-way automata
Math Systems Theory 4 1970 pp 24-28

B20 J HARTMANIS J E HOPCROFT
An overview of the theory of computational complexity
J ACM 18 1971 pp 444-475

B21 J HARTMANIS R E STEARNS
On the computational complexity of algorithms
Trans Amer Math Soc 117 1965 pp 285-306

B22 F C HENNIE
One-tape, off-line Turing machine computations
Information and Control 8 1965 pp 553-578

B23 F C HENNIE
On-line Turing machine computations
IEEE Trans on Electronic Computers EC-15 1966 pp 35-44

B24 F C HENNIE R E STEARNS
Two-tape simulation of multi-tape Turing machines
J ACM 13 1966 pp 553-556

B25 P MARTIN-LOF
The definition of random sequences
Information and Control 9 1965 pp 602-19

B26 A R MEYER D M RITCHIE
The complexity of loop programs
22nd National Conference Association for Computing Machinery 1967 pp 465-69

B27 D PAGER
On the problem of finding minimal programs for tables
Information and Control 14 1969 pp 550-54

B28 M O RABIN
Degree of difficulty of computing a function and a partial ordering of recursive sets
Technical Report 2 Hebrew University Jerusalem Israel 1960

B29 M O RABIN
Real time computation
Israel J Math 1 1963 pp 203-11

B30 R W RITCHIE
Classes of predictably computable functions
Trans Amer Math Soc 106 1963 pp 139-73

B31 R W RITCHIE
Classes of recursive functions based on Ackermann's function
Pacific J Math 15 1965 pp 1027-44

B32 A L ROSENBERG
Real-time definable languages
J ACM 14 1967 pp 645-62

B33 W J SAVITCH
Relationships between nondeterministic and deterministic tape complexities
J Computer Syst Sci 4 1970 pp 177-92

B34 R E STEARNS J HARTMANIS P M LEWIS II
Hierarchies of memory limited computations
IEEE Conference Record on Switching Circuit Theory and Logical Design 1965 pp 179-90

B35 E G WAGNER
Bounded action machines: Toward an abstract theory of computer structure
J Computer Syst Sci 2 1968 pp 13-75

B36 H YAMADA
Real-time computation and recursive functions not real-time computable
IRE Trans on Electronic Computers EC-11 Corr ibid EC-12 1963 p 400

B37 P R YOUNG
Toward a theory of enumerations
J ACM 16 1969 pp 328-348

Analysis of algorithms

Topics

1. Matrix multiplication in fewer than n^3 steps. Some lower bounds for matrix multiplication and inner product.[C3,C10,C13]

2. Polynomial evaluation. Optimality of Horner's rule in general case, preconditioning. Evaluation of a polynomial at more than one point.[C7]

3. Lower bounds for addition and multiplication of integers. Methods for multiplying in fewer than n^2 bitwise operations. Fast Fourier transforms and their application to multiplication.[C5,C8,C11,C12]

4. Sorting and searching methods. Treesort, radix sorts, polyphase sorts. Lower bounds on the number of comparisons necessary. Finding the ith largest in a set of elements. Evaluation of binary searching, AVL trees.[C2,C4,C6]

5. Graph algorithms. Planarity testing, connected components of a graph, transitive closure.

6. Miscellaneous algorithms. Matrix transposing in a paging environment, garbage collection, g.c.d. of polynomials, power of an expression, etc.[C1]

References—Analysis of algorithms

A considerable amount of current material is available only in the form of course notes or technical reports. Such items have been omitted here although courses being taught in analysis of algorithms are drawing up to half of their material from such sources. Thus, the references below serve more to give the flavor of this area than to constitute any approximation to a complete set of source material. Presumably much of the current material will appear in standard journals within a year.

Many informal working papers in this area can be found in recent Proceedings of the ACM Symposia on Theory of Computing (available from ACM) and Proceedings of the IEEE Symposia on Switching and Automata Theory (available from the IEEE or the IEEE Computer Group). Also, a "Bibliography on the analysis of algorithms" has been compiled by Professor E. M. Reingold, Department of Computer Science, University of Illinois at Urbana. As of the time of writing it has not been published.

C1 A BRAUER
On addition chains
Bull Amer Math Soc 45 1939 pp 736-739

C2 C A R HOARE
Quicksort
Computer J 5 1962 pp 10-15

C3 J E HOPCROFT L KERR
On minimizing the number of multiplications necessary for matrix multiplication
SIAM J Appl Math 20 1971 pp 30-36

C4 R M KARP W L MIRANKER
Parallel search for a maximum
J Combinatorial Theory 4 1968 pp 19-35

C5 D E KNUTH
The art of computer programming, volume 2
Addison-Wesley Reading Massachusetts 1969 sections 4.3.3, 4.6.3, and 4.6.4

C6 D E KNUTH
The art of computer programming, volume 3
Addison-Wesley Reading Massachusetts to appear 1972 section 5.3

C7 V YA PAN
Methods of computing values of polynomials
Uspehi Mat Nauk 21 1966 pp 103-134 Russian English translation in Russian Math Surveys 21 1966 pp 105-136

C8 A SCHONHAGE
Multiplication of large numbers
Computing 1 1966 pp 182-196 (German with English summary) See Comput Rev 8 1967 review number 11544

C9 P M SPIRA
The time required for group multiplication
J ACM 16 1969 pp 235-243

C10 V STRASSEN
Gaussian elimination is not optimial
Numer Math 13 1969 pp 354-356

C11 S WINOGRAD
On the time required to perform addition
J ACM 12 1965 pp 277-285

C12 S WINOGRAD
On the time required to perform multiplication
J ACM 14 1967 pp 793-802

C13 S WINOGRAD
On the number of multiplications necessary to compute certain functions
Comm Pure Appl Math 23 1970 pp 165-179

Program schemata

Topics

1. Flowchart schemata. Interpretations of schemata. Herbrand Interpretations. Undecidability of halting and equivalence problems. Various notions of equivalence.[D4,D10]

2. Special classes of flowchart schemata: Ianov schemata, two-location schemata, nested-loop ('while') schemata. Translation of flowchart schemata to 'while' schemata.

3. Recursive schemata. Translation of flowchart schemata to recursive schemata. Recursive schemata with no equivalent flowchart schemata. Decidable properties of monadic recursive schemata. Equivalence methods for recursive schemata; truncation induction.[D5]

4. Models of parallel computation. Determinancy; deadlocks. Equivalence-preserving transformations; increasing and decreasing the 'parallelism'.[D3]

5. Correctness of programs, choice of predicates associated with locations in a program.[D2,D6,D9]

6. Formal definitions of program structure.[D1,D12]

References—Program schemata

Some papers concerned with program definition and programming semantics are included as well as those dealing directly with program schemata. As before, much current material has not made its way into the accessible literature.

D1 E ENGELER
Algorithmic approximations
J Computer Syst Sci 5 1971 pp 67–82

D2 R W FLOYD
Assigning meaning to programs
In Proc Symposia in Applied Mathematics 19 American Mathematical Society 1967 pp 19–32

D3 R M KARP R E MILLER
Parallel program schemata
J Computer Syst Sci 3 1969 pp 147–195

D4 D C LUCKHAM D M R PARK
M S PATERSON
On formalised computer programs
J Computer Syst Sci 4 1970 pp 220–249

D5 J McCARTHY
A basis for a mathematical theory of computation
In Computer Programming and Formal Systems North-Holland Amsterdam 1963 pp 33–70

D6 Z MANNA
The correctness of programs
J Computer Syst Sci 3 1969 pp 119–127

D7 Z MANNA R WALDINGER
Toward automatic program synthesis
Comm ACM 14 1971 pp 151–165

D8 R MILNER
Program schemes and recursive function theory
In Machine Intelligence 5 Edinburgh University Press 1970 pp 39–58

D9 D PARK
Fixpoint induction and proofs of program properties
Ibid pp 59-78

D10 M S PATERSON
Program schemata
In Machine Intelligence 3 Edinburgh University Press 1968 pp 18-31

D11 D SCOTT
Outline of a mathematical theory of computation
Tech Monograph PRG-2 Oxford University Computing Laboratory Programming Research Group 1970 24 pp

D12 D SCOTT
The lattice of flow diagrams
Tech Monograph PRG-3 Oxford University Computing Laboratory Programming Research Group 1970 57 pp

ACKNOWLEDGMENTS

The author gratefully acknowledges the material and suggestions provided by E. A. Ashcroft, M. J. Fischer, A. R. Meyer, J. I. Munro and E. M. Reingold.

Social science computing—1967-1972

by HUGH F. CLINE

Russell Sage Foundation
New York, New York

Social science computing continues to grow both in quantity and variety of problem applications. Increasing numbers of social scientists are using increasing amounts of computer time on an increasing array of machines. Social scientists have upgraded their competence in computing and produced a number of programming systems for use in their research and instructional activities. Each year more social science graduate students are trained in the use of computers, and more social science departments now have at least one faculty member who is competent in computing. Of course, all these developments did not occur in the past five years. Important preliminary steps were taken as early as fifteen years ago. But during the period 1967–1972, social science computing has made significant progress and is now recognized as a member of the computing community.

In addition to increasing the amount of use, the number of programming systems, and the level of competence, social scientists have also vastly improved the exchange of information among computer users. Five years ago most social scientists were completely ignorant of computing activities outside their own institution. Many graduate training centers had developed a set of standardized, second-generation computer programs for social science research, but very few of these were ever exported to other universities. This was partially due to the difficulties of producing machine-independent programs, but patriotic zeal for the "local product" also created a reluctance to learn about other programs or systems. Many of these locally developed programs became obsolete with the distribution of third-generation machines, and social scientists, who were unwilling to make the effort to redesign and reprogram, took their heads out of the sand and began to look elsewhere. The transition from the second to the third generation caused many problems for social scientists as well as others, but it did lift the self-imposed intellectual quarantine among social scientists and created the climate for effective exchange of information on computing activities.

The most notable development in this regard has been the establishment of the *SIGSOC Bulletin*, a quarterly publication of the Special Interest Group for Social and Behavioral Science Computing of the Association for Computing Machinery. Under the editorship of John Sonquist, University of California, Santa Barbara, the *SIGSOC Bulletin* has become a major source of communication among the 450 subscribers. In addition, the activities of such organizations as the Inter-University Consortium for Political Research at the University of Michigan, the Roper Public Opinion Research Center at Williams College, and the National Program Library/Central Program Inventory Service for the Social Sciences (NPL/CPIS) at the University of Wisconsin have all contributed to the production of the new social scientist computer cosmopolite.

Despite all these promising developments over the past five years, it is not correct to state that social science computing is problem free. There are a number of very serious problem areas which require concerted attention in the next five years. In order to highlight these questions, I will present some findings from a study of computer uses in the social sciences which I have undertaken for Russell Sage Foundation. The study includes a description of current social science computing activities and an examination of the influence of computing on substantive social science problems.

Before turning to the results of the study itself, it should be useful to describe briefly in what ways social scientists are using computers. I have found it convenient to distinguish among four general types of computing applications: data processing and statistical analysis; simulation; computer-controlled experimentation; and large data file management. The most common application, data processing and statistical analysis, has often been referred to as the "bread and butter" of social science computing; for it typically accounts for the bulk of social science department computing budgets. Although most such applications involve numerical data, the analysis of textual materials is increasing. Typically, the investigator first puts the raw data into machine-readable form, i.e., on punched cards or magnetic tape; and then the computer is used for error detection, correction, and data reduction. Subsequently, canned programs, i.e., previously written general-purpose programs, are used to perform the more standard techniques of statistical analyses. For the most part, these applications of computers involve procedures which previously were accomplished on punched-card equipment and desk calculators. Although other types of computer applications in social science research will undoubtedly increase in the near future, data processing and statistical analyses will continue to be the modal type of computer usage for some time to come.

Simulation, the second type of computer use, is considered by many to be the most promising. Social science simulation is the construction and manipulation of an abstract operating model of a social system or process. Physical operating models such as wind tunnels and flotation tanks are well-known in scientific research, but all social science models of systems are abstractions stated in mathematical equations. The investigator designs his model by breaking a system down into subunits and describing both the subunits and the relationships between subunits in quantitative terms in a simulation programming language. The computer is then used to provide artificial blocks of time in which the system operates and subunits interact and are modified according to the model. The investigator usually compares initial and final values describing system subunits. During the course of the research, different initial values and relationships between subunits may be tried in the model to produce a better fit to data observed from the real world. Simulations have been done on many different types of problems with varying degrees of success including such things as racial integration in suburban housing tracts, old age pension programs, federal aid to state education programs, and international relations problems. Of course, the value of a simulation depends upon the investigator's ability to break a system into its subunits and realistically state initial values and relationships between subsystems.

The third type of computer use in social science research, computer-controlled experimentation, is for the most part still limited to the discipline of psychology. Real-time computers are being used by a small number of innovators to control, monitor, and change laboratory conditions. The feeding schedules for animals, the timing and sequencing of presentation of stimuli to subjects, and the monitoring and recording of communications between subjects in experimental settings are all now being done with the assistance of computers. In contrast to the other types of social science computer use, computer-controlled experimentation usually requires a machine that is used exclusively for this purpose. These computers are usually smaller and slower, but they require special-purpose input/output devices which are often unique and expensive. Using computers to monitor, modify, and in some cases to supervise research certainly will open new areas for social science research.

The use of computers for large data file management, the fourth type of application, is a new phenomenon in social science research. Many investigators are now using very large data files, such as the one-in-a-thousand sample of the census which contains over 15 million data items. It is virtually impossible to use these files for research without the aid of computers, and the problems of manipulating large data sets in an efficient manner are gaining increasing attention. Alternate methods of formatting and storing data files to match the input requirements of the standard analysis routines and new programs for large data files are being developed. In addition, social scientists are continuing to merge copies of sample surveys for secondary analyses. The problems of storing, searching, and retrieving subsets of data from these archives are now being handled on computers.

The survey of social science computing activities was limited to academic settings. Although it is quite true that important innovations in social science computing have occurred in independent research institutes, such as The Brookings Institution, the National Bureau of Economic Research, and The RAND Corporation, as well as in the research departments of large business corporations, it was decided to restrict the study to universities; because obtaining a comprehensive list of non-academic research institutes would have been a major research undertaking itself. The survey data were collected at the midpoint of the half decade, May 1969, with a four-page questionnaire mailed to the chairmen of the 517 departments which have granted at least one Ph.D. degree since 1960 in the following disciplines: anthropology, economics, agricultural economics, political science, psychology, educational psychology, and sociology. After two follow-up mailings and telephone calls, 421 completed questionnaires were received, giving an overall response rate of 81.4 percent. Table I presents the response rates for the disciplines, which range from 75.0 percent to 92.6 percent.

The 517 departments are located on 134 campuses. At least one department responded from 130 campuses; and this means that we have some data on social science

TABLE I—Disciplines and Response Rates

Disciplines	Questionnaires Mailed	Questionnaires Returned	Response Rate
Anthropology	47	36	76.6
Economics	99	78	78.8
Agricultural Economics	27	25	92.6
Political Science	95	80	84.2
Psychology	121	97	80.2
Educational Psychology	36	27	75.0
Sociology	92	78	84.8
	517	421	81.4

computing from 97.0 percent of the universities included in this study. Although the questionnaires were mailed to the chairmen, the cover letter encouraged them to seek the advice and assistance of their colleagues in answering the questions. The data should therefore be interpreted as organizational, i.e., department responses. This is appropriate, for the items were designed to collect information on computing activities and not the opinions or attitudes of any particular individual. The questionnaire elicited information on hardware, software, instructional programs, and levels of computing competence.

Turning first to hardware, Table II shows the proportions of departments reporting access to punched-card equipment and desk calculators. The first column headed "Department" reports the proportion of departments that had such equipment available for their exclusive use, and the second column gives the proportion of departments which had access to the machines in the central university computer center.

This data indicate that keypunches, sorters, reproducers, and card printers were more widely available in university computing centers and that desk calculators and programmable calculators (Wang, Monroe, Olivetti, and Marchang) were more frequently available for exclusive departmental use. Despite the fact that departments had somewhat limited punched-card facilities, most did have access through their computer centers.

TABLE II—EAM Equipment

Machines	Departments	Computer Center
Keypunches	41.6	85.0
Counter/Sorters	24.7	82.7
Reproducers	9.7	75.1
Card Printers	3.6	67.5
Desk Calculators	87.4	35.8
Programmable Calculators	15.9	1.2

A total of 358 computers were reported available for social science research and instruction. This is an average of 2.8 computers per campus. (These figures on number of computers are combined from department responses for each campus, and machines reported by several departments are counted only once.) Table III presents the distribution of computers by manufacturers reported in the study.

IBM computers accounted for 66.5 percent of the total. Approximately 50 percent of the IBM machines were third-generation, i.e., System 360; and about half of the third-generation machines were Models 50 or larger. Digital Equipment Corporation computers were the second largest type available to social scientists, but this manufacturer and all others were far below IBM in social science computing. Of all reported computers, 16.4 percent were DEC machines; and about 60 percent of these were PDP 8's. The Control Data Corporation was the third largest manufacturer with 7.8 percent of all the computers reported, and almost half of the CDC machines were 6000 series computers. Finally, UNIVAC was the fourth largest manufacturer with 3.1 percent, and about half of these machines were UNIVAC 1108's. All other manufacturers were under 2 percent, and these included Honeywell, Burroughs, General Electric, Xerox Data Systems, RCA, and Philco.

TABLE III—Computers by Manufacturers

Manufacturer	Number of Computers	Percent
International Business Machines	238	66.5
Digital Equipment Corporation	59	16.4
Control Data Corporation	28	7.8
UNIVAC	11	3.1
Honeywell	6	1.7
Burroughs	5	1.4
General Electric	4	1.1
Xerox Data Systems	4	1.1
RCA	2	0.6
Philco	1	0.3
Total	358	100.0

Dominance of IBM in social science computing is of course no surprise. However, the fact that two-thirds of the computers available to academic social scientists were manufactured by one supplier should not be interpreted as widespread uniformity in social science computing systems. Over 22 different types of IBM

TABLE IV—IBM Computers

IBM Model	Number of Computers	Percent
1130	25	10.5
1401	35	14.8
1500	2	0.8
1620	22	9.2
1710	1	0.4
1800	8	3.4
7040	6	2.5
7044	3	1.3
7072	2	0.8
7700	1	0.4
7094	16	6.7
360/20	16	6.7
360/25	1	0.4
360/30	8	3.4
360/40	22	9.2
360/44	5	2.1
360/50	27	11.4
360/65	15	6.3
360/67	9	3.8
360/75	11	4.6
360/91	3	1.3
Total	238	100.0

machines were reported. Table IV shows the distribution of IBM machines by model.

These machines vary widely in cost, instruction sets speed of the central processing units, size of memories and available input/output devices. This variation shows very clearly why it has been extremely difficult to produce programs which can be widely used on many different types of machines.

The average number of computers per campus was 2.8. This figure is somewhat misleading, for these computers were not evenly distributed across all universities or departments. It is often the case that members of one department have access to a computer; but due to funding or administrative restrictions, members of other departments on the same campus may not have access. Table V lists the distribution of the number of computers reported available to the 421 departments.

Notice this distribution is quite uneven. The modal category of one computer includes 35.9 percent of the departments. A very small proportion, 3.8 percent, of the departments were comparatively quite wealthy, having access to six or more computers; and 28 departments, or 6.6 percent, reported they still did not have access to any computer.

In addition to the type and number of computers available, the location and administrative auspices of the machines also influence their utility for research and instruction. Almost 70 percent of the computers were located in a central university computing center. The remaining 30 percent were widely distributed among other university computer centers, research institutes, departments, commercial service bureaus, and combined social science department computing centers. Despite the many recent debates over the advisability of centralization or decentralization of university computing facilities, the data show that the pattern of centralization had not changed very much; and only in a very small proportion of cases, 5.3 percent, was there a computing facility for the exclusive use of the social scientists.

University-based scholars in many disciplines have recently expressed enthusiasm for the possible applications of time-sharing systems in their research and instruction. Some type of time-sharing system was reported at 33, or 25.4 percent, of the universities included in this study. Most of these were interactive compilers, which can be used for finding errors in programs but not for data processing and analysis. Only ten universities had fully operational time-sharing systems, which can execute programs; and most of these had severe restrictions on the size of the data files they can handle. In addition, 29 universities reported using some type of commercial time-sharing service. It is not surprising that time-sharing has not been more widely used in the social sciences, for very few of the existing systems are capable of efficiently handling large input or output files. Since many social scientists typically have medium- to large-sized data files, they find that most existing time-sharing systems simply cannot be used.

Regardless of whether a computer is used in a time-sharing or conventional manner, the hardware, including the central processing unit, the memory, and the input/output devices must be provided with the software or programs to execute the symbol manipulations the scientist requires for his problem. Although software usually refers to operating systems, compilers, assemblers, and applications programs, this study of com-

TABLE V—Number of Computers Available to Departments

Number	Percent
None	6.6
1	35.9
2	29.0
3	13.8
4	6.2
5	4.7
6+	3.8
	100.0

puter uses in the social sciences has been limited to higher-level languages and packages of programs. Table VI presents the higher-level languages most commonly reported by the departments.

FORTRAN IV comes close to being a universal language, but the fact that 98 percent of the departments with access to computers reported the availability of FORTRAN IV does not automatically produce uniformity and standardization, for there are many variations of the language. Limited instruction sets and maximum permissible program sizes are two of the major sources of variability. The languages most commonly available are those which the major manufacturers support with continuous error correction, documentation, and further development. The fact that FORTRAN and COBOL are so widely available to social scientists is for the most part a result of IBM's policy to support these two languages on most of their computer systems.

TABLE VI—Languages Available

Language	Percent
FORTRAN IV	98
COBOL	77
ALGOL	62
PL/I	51
FORTRAN II	49
WATFOR	44
SNOBOL	38
BASIC	28
OTHER	55

Over ninety different languages were reported in this study. Some of these have been developed locally at the universities, and others have been produced by small, commercial software companies. One can quite legitimately question the extent to which these languages are thoroughly debugged and documented, for very few universities or commercial software houses have the manpower and expertise necessary for the adequate support of languages.

The distribution of higher-level languages available to social scientists is more easily understood by examining the number of languages reported by each department, as shown in Table VII.

TABLE VII—Number of Languages

Number	Percent
1-2	10.5
3-4	23.6
5-6	29.9
7-8	18.9
9+	17.1
	100.0

The average number of languages reported by each department was 5.9. The typical pattern for a department was to have two or three languages maintained by the hardware supplier and several additional, special purpose languages maintained locally. It should be pointed out that these data refer to the number of languages reported *available* to social scientists. In a mail questionnaire, it was not possible to assess the extent to which these languages were used. However, on the basis of personal visits to over a dozen major social science graduate training centers, it is my clear impression that most social scientists use a subset of the available languages. Many social scientists adopt one and only one computer language, and they sing praises to this language with the same intense emotional commitment as the immigrant who extols the virtues of his new country.

Departments were also asked to report on the data processing and statistical analysis packages of programs available to the members of their departments. The distribution of the most commonly reported packages is listed in Table VIII.

The BMD Package, developed under the direction of Professor W. J. Dixon at the Health Sciences Computing Facility, University of California, Los Angeles, comes closest to being the universal social science statistical package.[1] Of the departments using computers, 76 percent report having access to the BMD Package. The various computer programs included in this package have proved useful in the past in many types of biomedical research projects at UCLA. Written in FORTRAN IV, and therefore relatively easy to export to other computer systems, the programs in this package include routines for data description and tabulation, multivariate analyses, regression analyses, time-series analyses, variance analyses, and a series of special programs for data transformation and scaling. Although

TABLE VIII—Packages Available

Package	Percent
BMD	76
SSP	52
SPSS	16
OSIRIS	16
Data-Text	13
Other	87

created in response to the needs of biomedical researchers, this library of programs has been more widely distributed to social scientists than any other package.

The second most widely available statistical package was the Scientific Subroutine Package (SSP).[2] It is a collection of over twenty different statistical routines maintained and distributed by IBM. Because it is written in FORTRAN IV, it is also easily exported. However, the Scientific Subroutine Package is not easily used by most social scientists. As the name implies, this package is a collection of subroutines; and the user must write a FORTRAN main or calling program for the subroutine he requires. Very few social scientists are able to write a FORTRAN program with appropriate linkages to subroutines. Despite the fact that our data show that 52 percent of the reporting departments had access to the Scientific Subroutine Package, I seriously doubt the extent to which this package is heavily used by social scientists.

The remaining three packages most commonly reported by our respondents were the Statistical Package for the Social Sciences (SPSS) developed by Norman H. Nie of the National Opinion Research Center and University of Chicago,[3] the statistical package OSIRIS developed at the Institute for Social Research at the University of Michigan, under the direction of Ralph Bisco,* and Data-Text, developed by Arthur Couch and David Armor at Harvard University. SPSS and OSIRIS are both written in FORTRAN IV and are relatively easy to export. The Data-Text Package was originally written in FAP, the assembly language of the IBM 7094, and operates only on the few remaining 7094 installations. Under Armor's direction, Data-Text was being reprogrammed in FORTRAN IV, and an expanded version was distributed in 1971.[5] Both Data-Text and SPSS implemented a user-oriented language in which job parameters are specified in a format similar to the natural language social scientists use in discussing data processing and statistical analyses. For example, one requests cross-tabulations by preparing a control card which states: COMPUTE CROSSTABS SEX BY OCCUPATION. OSIRIS has recently substituted for numerical job parameters a similar user-oriented language.

Eighty-seven percent of the departments reported packages other than the five mentioned above, and there were over 170 different packages. One can, of course, question the extent to which these are completely different packages. In many cases, universities have made minor modifications in existing packages and merely given them a local name. Nevertheless, it has become a common practice for many universities, and in some cases even for departments, to develop a set of programs which have a common input/output format. These packages are locally used and maintained. Typically the documentation is exceedingly poor. Efforts to support these programs are not well funded; and most of the programs have never been thoroughly debugged.

Despite the fact that over 170 packages were reported by the departments, the number of packages available to each department was quite small. Table IX lists the distribution of the number of packages.

The average was 3.0 packages per department, and the distribution was highly skewed in the direction of one, two, and three packages per department. Given the problems of importing, documenting, and maintaining these packages, it is not surprising that the number is quite small. Furthermore, since most of these packages include similar routines, there is very little to be gained by increasing the number of packages available.

Both the excess of locally developed packages and the small number of packages available to any given department point to some of the major problems in social science computing. Firstly, many of the packages leave a great deal to be desired in terms of accurate and efficient algorithms. Furthermore, a social scientists who develops expertise in using only local packages typically finds himself completely ignorant when he moves to another university.

Finally, and most importantly, the distribution of existing statistical packages has significant implications for substantive social science research problems. Despite the fact that most departments have access to more than one package, it is my clear impression that social scientists still prefer to use the locally developed package. Drawing from either a sense of comfort in using the package developed by someone they personally know or

TABLE IX—Number of Packages

Number	Percent
1	24.3
2	25.4
3	21.7
4	9.5
5	9.0
6	4.6
7+	5.5
	100.0

* Documentation for OSIRIS is maintained in machine-readable form for frequent updating and is available upon request from the Institute for Social Research, The University of Michigan.

from some sense of pride in "our thing," most social scientists avoid using the more widely distributed packages. This preference for the local package becomes a more serious problem when it is limited in the number of data analysis routines offered to the user. For example, many of the local packages only have routines for cross-tabulations. This means that both students and faculty are indirectly encouraged to conceive all their substantive problems in a cross-tabulation research design.

Despite the excessive duplication of insufficient efforts in producing packages, there are still many universities that are attempting to design new packages for data processing and statistical analysis. Clearly there is a need for some duplication and competition; but what seems more critical now is leadership in establishing some cooperation, uniformity, and standards in the development of these packages.

Of course, the availability of adequate hardware and software is not sufficient for solving the problems of effectively using computers in social science research and instruction. Most social scientists have become painfully aware of the fact that it is necessary to develop some competence in computing to use these machines in pursuing substantive research problems. In order to assess the extent to which social scientists are gaining some competence in computing, the respondents were asked to indicate separately for undergraduates, graduates, and faculty members the proportion who: (1) can write programs in higher-level languages; (2) can use statistical package programs; and (3) use computers in research problems. Table X lists the average proportions reported.

Among the undergraduates, only 15.2 percent could write programs in higher-level languages; 18.8 percent could use statistical packages; and 16.4 percent used computers in research. For graduate students the figures are slightly higher. Roughly one-quarter, 23.6 percent, knew a higher-level language; almost one-half, 41.3 percent, could use statistical packages; and slightly over one-half, 50.6 percent, used computers in their research. On the other hand, faculty members really possessed no more competence in writing programs than undergraduates, 16.9 percent. Fewer faculty, 32.7 percent, than graduate students were able to use statistical packages. Finally, just under half, 48.7 percent, of the faculty members were reported as using computers in their research.

TABLE X—Levels of Competence

Proportion of Individuals Who	Percent		
	Undergraduate	Graduate	Faculty
Can Write in High Level Languages	15.2	23.6	16.9
Can Use Statistical Packages	18.8	41.3	32.7
Use Computers in Research	16.4	50.6	48.7

Without getting into the old argument as to whether or not the social science disciplines should be using quantitative analyses in their research, it is still quite interesting that in these disciplines, where at the most basic level research involves symbol manipulation, only one-half of the faculty members are currently using computers in their research activities. In addition, the graduate students are more competent than the faculty. At the minimum, one can conclude from these data that social scientists have been slow in building computer competence in their disciplines.

It is often the case that effective use of computers in social science research depends upon the availability of at least one individual who can serve as a liaison between his colleagues' substantive problems and the computer. This individual can frequently be of great assistance in preparing data and programs for computer processing. In an effort to assess the extent to which such individuals were available, the chairmen were asked a series of short questions concerning the computing skills of the most competent member of the department.

Only 26.1 percent of the departments report that they have someone who is able to instruct himself on the use of statistical packages; only 21.4 percent of the departments report the availability of anyone who is able to write computer programs; and only 18.1 percent of the departments report one member who can interpret error messages and memory dumps to determine the reasons for unsuccessful runs. This very low report of computer competent colleagues accounts for some of the difficulty which many social scientists experience in trying to use the computer in their research.

Given this low level of computer competence, one could expect some evidence of concerted attempts to introduce new instructional programs to produce scholars with some training in computing skills. The chairmen of the departments were asked to indicate separately for their undergraduates, graduate students, and faculty members, the options available for obtaining instruction in computing. The proportion of departments which reported the different ways in which members can learn about computing is listed in Table XI.

The data show clearly that most of the computer instruction given to social scientists at all levels comes

TABLE XI—Computer Instruction

Instruction Available	Percent		
	Undergraduate	Graduate	Faculty
Department Computer Course	7.9	12.6	7.6
Department Research Course	4.6	19.8	4.9
Other Department Computer Course	47.0	27.5	16.6
Non-credit Computer Center Course	31.8	31.4	50.3
Self-instruction	2.6	2.9	9.7

either from other departments or the university computer centers. This is reminiscent of the debate popular some years ago on the question of who should teach social scientists in statistics courses. The consensus on that issue now seems to be that the instructors should be individuals who know social science research. It seems unlikely that the staff of other departments or computer centers will have this knowledge. Yet, very few departments taught undergraduates any computing in courses taught within the department. A slightly higher but still disappointingly low proportion of the departments reported that their graduate students learned in either a departmental computing course or a departmental research course, 12.6 percent and 19.8 percent respectively. Both the non-credit computer center courses and courses taught in other departments were the most common options available to all social science students. The most common option for faculty members, on the other hand, was only the short non-credit course at the computer center. These courses are typically crash courses which either teach one higher-level language, most commonly FORTRAN, or the details of job submission and operating system control languages for that particular computing facility.

Given the problems noted above concerning the availability of computer hardware and software for the social sciences, it also appears that there is a critical shortage of individuals who understand both substantive social science problems and computing. A very low proportion of departments reported that undergraduates, graduates, or faculty members are learning about computing either within their own departments or elsewhere, and apparently very little is being done to solve the manpower problem. Graduate students are the most competent group of social scientists in computing, and even they are severely limited to the use of packages in pursuing substantive problems.

Although much progress has been made in social science computing in the past five years, the data from this study indicate that there are a number of major problems which severely restrict more effective computer-assisted research. These include providing more equal access to adequate hardware, decreasing the number of packages under development, and providing better support for the documentation and debugging of the more widely used packages. In conclusion I should like to point to two additional problems, whose successful solution I consider as critical.

Perhaps the most immediately pressing problem is the low level of competence. Social scientists must insure that adequate instructional programs are offered in all graduate departments, and students should learn more than simply how to use the locally developed software. I would argue that it is irresponsible to produce professional social scientists today who will have an active career in social science research for the next forty years without at least some introduction to the basic design and logic of computers.

A second very pressing problem which is still not satisfactorily resolved is that of compatibility. It is still true that most computer programs and data files cannot easily be exchanged among social scientists using different computers. Very few of the early pioneers in social science computing appreciated the need for preparing programs and data files for export to colleagues using different computers. Today, the problem of exportability has reached staggering proportions. Individuals find that many programs and data prepared for initial processing on one computer simply cannot be run at another computer center which uses exactly the same model and manufacturer. Local variations in peripheral devices and programming procedures make it extremely difficult and sometimes impossible to move from one computer to another.

Although one may place the blame upon the designers of the early computing systems for not having the foresight to think in terms of standardized data files and programming conventions, or one may cite the commercial manufacturers for not establishing industry-wide standards, the problem still remains; and social scientists cannot now realistically look to others for solutions. They must among themselves agree upon common standards for program design and data file preparation which will minimize the problem of exportability. Granting agencies, both public and private, which support the development of social science computing can be of great assistance in insisting upon adherence to standards; but the original impetus for this movement must come from the social science community.

REFERENCES

1 W J DIXON ed
BMD biomedical computer programs
Berkeley Cal University of California Press 1970

2 IBM Manual GH20 0166
Scientific subroutine package—Version III

3 N H NIE D H BENT C H HULL
Statistical package for the social sciences
New York McGraw-Hill 1970

4 A S COUCH
The data-text system
Unpublished user's manual Harvard University 1965

5 D J ARMOR A S COUCH
The data-text primer
New York Free Press 1972

Future developments in social science computing

by GEORGE SADOWSKY

The Urban Institute
Washington, D.C.

INTRODUCTION

The set of computer related activities characterized by the term "social science computing" is both diverse and extensive. During the past 15 years, such activities have grown substantially both in scope and in volume and have become increasingly important both for basic research in the social and behavioral sciences and for public policy formation and evaluation.

In this paper I will present an overview of the evolution and present status of social science computing and examine several factors that are likely to have a significant effect upon its development during the next 10 years. Predicting the future is generally hazardous and is complicated in this instance by being dependent upon the development of future computing technology. During the relatively short history of automatic digital computing there have been many predictions—both optimistic and pessimistic—that have returned in short order to plague those who made them. Despite the high risk of error, attempts to predict the future have some value in planning for that future state. Attempts to forecast the future also can often provide a more accurate assessment of the present, a process from which social science computing might benefit considerably.

The term "social science computing" is somewhat misleading in that it implies a homogeneity of computing activities that does not now exist. Activities as diverse as process control in computer controlled experimentation laboratories, dynamic macroeconomic simulation, large survey data collection and analysis, simulation of alternative social policies, dissemination of census results, multivariate statistical techniques, and natural language text processing for content analysis and concordance production are all included. Social science computing activities may be observed as a part of research activity in universities and research institutions, in commercial organizations that produce social science computing products and in those that use them, and in government administrative processes and policy evaluation functions. Statements regarding social science computing generally are true of a subset of these activities only. To the extent that there is a commonality among activities in the field, it is that they are a collection of processes centered upon observed data relating to behavioral phenomena.

Many aspects of social science computing enjoy an independent tradition. The methodologies of commercial data processing, information retrieval, source data capture, statistical inference and digital system simulation all apply to social science computing activity, yet professional and methodological development in these fields generally occurs independently of it. The dependence of social science computing upon a variety of existing methodological traditions reduces its cohesiveness as a discipline and complicates the process of training social scientists for research careers in which a knowledge of quantitative methodologies is becoming increasingly important.

HISTORICAL OVERVIEW

Social science automatic computing activity dates from the introduction of automatic data processing equipment in the 1880s by Herman Hollerith. Hollerith's machinery was initially used in 1887 by the City of Baltimore to tabulate mortality statistics,[17] and was later used by other local governments for similar functions. However, its use in sorting and tabulating the results of the 1890 U. S. Decennial Census of Population marks the beginning of large scale social science computing. Hollerith's initial innovations led to the development of a family of "unit record" electromechanical accounting machinery for general data processing applications. A primary use of these machines was to collect, process, and disseminate statistics for public purposes. The introduction of Hollerith's automatic machinery in the census process was followed by a period of increasing use of automatic accounting machinery by government for processing social and economic statistics. Shortly after the production of electronic digital computers, the first computer produced commercially—a Remington Rand Univac I—was installed at the U. S. Bureau of the Census in 1951 for processing data collected during the 1950 U. S. Decennial Census of Population.[19]

The automation of statistical computing activities began well before the invention of the automatic digital computer. Many statisticians made intensive use of the commercial unit record equipment available prior to the digital computer. Despite the fact that calculations performed using this equipment were slow, required extensive wiring of plug boards, and required repetitive use of different specialized machines, the resulting procedures were generally more accurate and faster than the corresponding calculations performed by hand. Thus statisticians quickly took advantage of the opportunities offered by automatic digital computation.

During the initial years of automatic digital computer use, social science computing tasks were similar to many other computing tasks. Computers were programmed almost exclusively in primitive assembly languages, the programmer often operated the computing equipment himself, and computing activity was localized around the few computer installations then available. Survey research centers and individual social scientists analyzing empirical data continued to rely heavily upon unit record processing equipment, especially specialized equipment such as the IBM 101 Counter-Sorter. The development of higher level programming languages such as Fortran and Comtran allowed programmers to write programs that were more easily understandable by non-programmers and more easily exportable to colleagues using different computing equipment. During this period the initial "general purpose" statistical and data processing programs were produced; such programs were often referred to as "canned" programs or "program packages." They were characterized by being machine dependent, having numeric field control cards and fixed field punched cards as input, and producing a limited range of outputs, usually sparsely labelled. Substantial emphasis was placed upon internal execution efficiency.

An initial milestone in the development of statistical computing was achieved when W. J. Dixon and his colleagues at UCLA created the original set of biomedical statistical programs[6] in 1962. This set, initially known as the BIMD series, provided the user with an integrated set of programs containing a large number of statistical procedures; the series included such features as a common vocabulary across programs, a comprehensive method of specifying data transformations, adequate user documentation, and readable and intelligible printed and graphic output. The BIMD series surpassed previous products by its completeness, increased user orientation, and emphasis upon exportability to other computer centers. While some aspects of the original BIMD programs now appear primitive, the initial impact of this set of programs upon statistical computing was substantial. The BIMD programs were developed during a time of intense batch program construction activity; some of the batch systems that followed BIMD such as P-STAT[2] and DATA-BANK-MASSAGER[13] were oriented specifically to social science users.

Another milestone in the development of social science computing was the creation and implementation of higher level languages oriented specifically to tasks in the field. Shortly after the concept of higher level languages became generally accepted, a number of social scientists began to develop languages oriented specifically to social science computation. I believe primary credit should go to Arthur Couch and David Armor at Harvard University for their development of the Data-Text language.[8] Although implemented originally in assembly language, Data-Text provided social scientists with free form statement types linked directly to behavioral science computing procedures. Considerable effort was spent upon organizing printed output to the user's specifications. The development of Data-Text led to other efforts such as BEAST[1] and SPSS[16] which have been useful as system design experiments and as applications programs for obtaining statistical outputs and tabulations from rectangular data files.

A third milestone in social science computing was the successful exploitation of interactive computing systems for research and instruction in the social sciences. Using computer systems that support economical interactive computing, a number of such programs have been developed. These include IMPRESS[15] for on-line instruction in social science computer based research methods, BASIS[20] for on-line statistical analysis, TROLL[12] for on-line time series analysis, equation system estimation, and econometric simulation, and others such as TRACE-III,[3] OPS-3,[7] and ADMINS.[14] These systems are examples of the manner in which interactive environments have been used to support powerful tools for social science computing tasks.

Most social science computing activity has depended critically upon access to behavioral data in machine readable form. Government agencies and survey research organizations pioneered in developing techniques for recording survey data on punch cards and analyzing these data mechanically. Universities and private survey research centers today have substantial holdings of sample survey data, and much of this material is available for secondary analysis. Several specialized data distribution networks have evolved; for example, the Inter-University Consortium for Political Research at the University of Michigan distributes political data by mail from its archive to over 100 member academic organizations, and the National Bureau of Economic Research makes available its on-line data bank of aggregate economic time series to its subscribers.

Government statistical agencies have historically been important sources of data for quantitative social science research. The publications of these agencies were an important source well before the widespread availability of automatic computing machinery. The U.S. Bureau of the Census provided a wide variety of population, housing, labor force and manufacturing data on an aggregated basis for many years; and after the 1960 census the Bureau initiated a policy of disseminating population microdata on magnetic tape for 1/10000 and 1/1000 random samples of U.S. families. Since 1969, microdata collected by the monthly Current Population Survey have been available in machine readable form to qualified researchers. Availability of public data in machine readable form has increased in recent years as government statistical operations have become more mechanized and as social scientists have learned to obtain and use the data effectively.

CURRENT STATUS

Social scientists have available to them today a wide variety of computer programs for their use. Batch-oriented programs for statistical analysis of rectangular file structures are most common; in addition to the better known and more reliable of these programs, there exist many locally developed programs that perform frequently used statistical procedures. The quality, accuracy and documentation of these programs vary substantially, and recent evidence presented by Longley[11] and Wampler[21] indicates that increased quality control is needed in their production. Computing languages for the social sciences are relatively new; however, there has been sufficient development to indicate the power that such languages provide, and their continued development seems warranted. More recently, interactive programs have been developed that give social scientists flexible control over their procedures and their data. Some very useful interactive tools have already been produced, but much work will be required to exploit current interactive computer systems in a truly productive manner.

An increasingly copious flow of behavioral data is now generated by individual researchers, private firms, public organizations and governmental organizations, and much of it is in machine readable form. The increase in complexity of the processing operations now being used require the input data to be more error free than was required in the past; present consumers of microdata especially must be much more sensitive to noise in their data. There is now increased demand for government data by private researchers, and the conflict between the use of individual data for legitimate public purposes and the confidentiality of such data has yet to be resolved in a satisfactory manner.

Major problems currently exist within the field of social science computing. Among the most serious appear to be: (1) inadequate standards for data documentation and problems of data transfer; (2) the low level of computational knowledge among social scientists and the lack of adequate training available; (3) the slow rate of diffusion of computing innovations into social science computing, especially in government; (4) problems of program inaccuracy, documentation and transfer; (5) lack of adequate software tools for many types of complex processing operations; and (6) the low level of professionalism in social science computing activities generally.

Some of the problems that exist in social science computing are common to most areas of computing and will be ameliorated by advances in computing generally. Other problems have been aggravated by the lack of resources allocated to the social sciences relative to other disciplines. Still others can be traced to misallocation of those resources made available. Finally, it appears that the current organization of the social science computing industry has itself produced or aggravated a number of present problems.

INDUSTRIAL ORGANIZATION

It seems both appropriate and relevant to employ one of the social sciences, economics, to study the industrial organization of the social science computing industry. For the totality of social science computing activity is an industry; it encompasses a variety of products, their producers and consumers, the markets in which they interact, the mechanisms for distributing goods, the reward structure for producers, the rate of investment in capital, incentives for investment, the rate of technical progress, and other industry characteristics. The institutional structure of social science computing directly affects the manner in which individuals and organizations allocate resources to produce a wide variety of products. Solutions to current problems in social science computing are as likely to result from organizational change as from technical advances or increased resource supplies.

Most products within the social science computing industry fall into four general groups: (1) computer programs; (2) numeric data files; (3) knowledge regarding numerical methods, statistical procedures and computing techniques; and (4) personal services. Within each of these groups, the products produced are heterogeneous, although there is some evidence of product differentiation that is largely artificial.

Computer programs range in generality from specialized applications programs to general purpose applications systems and translators for high level programming languages. The former are final goods; they represent unique products designed to satisfy specific requirements. General purpose programs are intermediate goods, or producer goods; once created, they are then used repeatedly to produce a variety of final goods specified by non-programming computer users. Data files vary in size and complexity from those having a rectangular structure with few observations and variables to those resulting from the 1970 Decennial Census of Population and Housing.

The market for social science computing products appears to have four important dimensions: (1) locality; (2) computer type; (3) substantive focus; and (4) profit/non-profit. Markets have a strong local component because computing installations serve as centers of knowledge and expertise for their users, and rapid and inexpensive access to assistance is often essential for the success of computer related work. In addition, an organization having a computer center within its administrative jurisdiction frequently raises barriers to use of other centers by their staff. Also, the market is divided according to the manufacturer, model, and configuration of computing equipment available because programs, data and technical information can often be transferred at low cost between quite similar computers but not between dissimilar computers. The market is also stratified by substantive social science discipline or research or policy focus, especially with regard to machine readable data files and computer programs and hardware configurations that perform specialized processes within that discipline. Finally, there is infrequent interchange of programs between commercial firms and academic and research institutions.

There are a large number of producers of social science computing products, many of whom are individual social scientists and programmers. In addition, producers include social science computing groups within research organizations and universities, survey research centers, private firms, and various agencies of federal, state, and local governments. Many of these producers are also consumers; often production is initiated primarily or wholly for internal consumption. Most producers are active primarily within one of the above markets, although private firms can generally respond quickly to changes in demand by acquiring and reallocating resources.

With one outstanding exception, the subsets of the industry corresponding to the above broad product groups exhibit a low degree of industrial concentration, i.e., no small number of producers command a major share of these sub-markets. There currently exist a large number of producers of social science software, and except for large applications systems barriers to entry into the industry are almost non-existent. The number of producers (suppliers) of data is smaller, and entry into this industry generally requires a higher level of investment than entry into program production. Several organizations have made an investment in data banks containing economic time series, and competition to provide these data and associated computer software is now vigorous. Government agencies enjoy a natural monopoly position in collecting and disseminating data when the data must be made available to the government by law and is unlikely to be disclosed otherwise; examples include population census information and individual and corporate tax return data. In addition, government agencies enjoy substantial competitive advantages in collecting such data when it results as a byproduct of on-going administrative procedures or when the resources required to produce the data are larger than most potential producers can mobilize.

The performance of government in producing data in such a quasi-monopoly position has not been without fault. Under monopoly conditions, economic theory would predict that prices would be higher than if competition existed and the single producer would earn monopoly profits because of it. Government agencies are generally prohibited from pricing their products above average cost at most; but instead the monopoly position appears to encourage inefficient use of resources and technology and a lack of responsiveness to consumer demands. This results in higher prices for the consumer than would have prevailed in a competitive environment, a sluggish rate of innovation and less consumer choice.

The distribution mechanism for social science computing products is in the process of substantial change. Products have been generally recorded on physical storage media such as punch cards, printed paper, and magnetic tape and these media were physically distributed to consumers. In the past, high costs associated with this distribution technology were partially responsible for product markets being segmented by computer installations and by hardware type. More recently, an increasing number of products are being distributed by transmitting digital data over common carrier telephone networks and specialized communications networks, and it appears that this growth will continue both in relative and absolute terms for some time.

Knowledge about social science computing products, statistical procedures, and related computing techniques reaches potential consumers in a number of ways. Inter-personal communication now forms one of the most important means of transmitting such knowledge. Some dissemenation of this information is made in publications of computer societies, social science professional societies, research institutions, and various related academic disciplines. Informal publications, workshops and conferences also play an important part in collecting and distributing knowledge. However, at the present time there are no information centers or professional societies focused primarily upon social science computing. SIGSOC, the Special Interest Group for Social and Behavioral Science Computing of the Association for Computing Machinery, is a likely candidate for this position, but its development into a professional society will take at least 5 to 10 years.

Social science computing products are created in accordance with the production processes available to the industry at the time of production. A production process or function is a set of technical rules that specify how goods and services called inputs may be transformed into other goods and services called outputs.[19] The levels of knowledge and technology available dictate what production functions are available for obtaining specified outputs. If a production process requires more than one input, there are usually a variety of combinations of inputs that are capable of producing the same set of outputs, i.e., the production function may allow input substitution. In computing, a typical example of input substitution is provided by the frequent tradeoff in program design between the use of main memory and processor time; using more of one generally provides savings in use of the other. Resource constraints and the relative prices of the inputs enter into the choice made by producers of which production process to use for obtaining output.

For many social science computing products, the cost of producing the first unit of a specific output is high relative to the marginal cost of producing subsequent duplicate units. The cost of duplicating a computer program or a machine readable data file, once the product has been created, is low and requires only the use of well-known duplication procedures. However, the costs paid by social scientists to import programs and data are generally much higher; the principal reason is that the distribution technology for these products is still rather primitive since the products often do not adapt easily to computing environments other than those very similar to the environment in which they were created. The total costs of distribution are therefore often much higher than the costs of duplication.

If the marginal cost of a unit of production is low relative to the initial investment required to produce the first unit, then resources are allocated more efficiently if there exist a few large producers rather than many small producers. Yet with the exception of the production and distribution of large data files and a few other products, it appears that the market is dominated by small producers. There are a number of factors that help to explain this phenomenon. First, many of these products are differentiated slightly but meaningfully for their consumers. Second, computing products may be one of a number of outputs of a joint production process; often another output is increased education for the producer. Also, the very low barriers to entry encourage new producers into these markets at low cost to the new entrants.

The above factors help to explain the existence of small producers of social science computing products, but they do not explain the relative absence of larger producers who are able to make substantial investments in more general and more powerful products. Even for the first unit of production, there appear to be substantial economies of scale. That is, there are some fixed costs in program preparation that must be incurred regardless of program size. Examples of such fixed costs in social science computing systems are code to read the data file and interpret its values, code to provide elementary output displays and code to perform standard data transformations; they are almost always included regardless of how the data are processed. To the extent that fixed costs influence the production process, a more comprehensive and more general program appears to be cheaper to construct than the many smaller programs that duplicate its functions.

A very important factor limiting substantial investment in computing products is that although the return to such investment may be high for the industry as a whole, most of the return cannot be captured by the investor. Another factor is the segmentation of the market discussed previously; if individual markets are limited in size, then the cost of sales increases and potential returns are more limited.

Much social science computing activity occurs in universities, research organizations, government agencies and other environments that often do not enter the commercial marketplace to buy and sell these products. Within these organizations, producers are generally individuals, and the rewards sought are primarily nonfinancial; rather they are prestige and recognition for the producer's accomplishment. At the present time, however, the technical activity associated with producing computer products for social scientists does not enjoy high prestige or substantial recognition for the individual involved. "Tool using," or substantive activity, has higher status within social science professions than "tool building," or technical activity. Thus, although changes in the level and allocation of investment in computing tools for social scientists would lead to a more efficient allocation of individual skills and technical resources and would be expected to provide substantial externalities to social science as a whole, the reward structure of academic social science disciplines prevents the individual innovator from collecting his rewards. The lack of a satisfactory reward structure for non-commercial producers of social science computing tools is very unfortunate, since it discourages innovative investment in production for those who are often able to make substantial contributions to the field. The inadequate reward structure contains another unfortunate aspect; it perpetuates a lack of interest and involvement in computer oriented training by persons in the social science professions.

THE FUTURE

During the next ten years, continued substantial growth in social science computing activity is likely to occur. Social scientists currently being trained are taught quantitative methods to a greater degree than any of their predecessors. The amount of data in machine readable form is increasing rapidly. The decreasing cost of computation is shifting the balance in favor of using computers rather than using manual labor or not performing a task at all. Furthermore, as inventories of data and quantitative methods grow and as the number of practitioners in social science computing increases, important issues in social and economic policy become more tractable using the computer. Although growth in social science computing will not be uniform and will be unevenly divided among disciplines and applications, there is little doubt that it will occur.

Advances in computing hardware and software during the past 15 years have been responsible for a dramatic decline in the cost of computing. Current evidence suggests that logic and memory costs will continue to decline substantially during the next decade. These declines in cost will benefit social science computing activity by reducing the cost of those computing inputs.

In addition to declines in computing cost, three specific areas of development within the computer industry are likely to benefit social science computing substantially and should be assisted and encouraged. These areas are: (1) communications terminal technology and interactive system development; (2) digital data networks linking computers; and (3) data storage technology. Developments in these three areas will impact different aspects of the social science computing industry.

The development of reliable and responsive interactive computing systems has already had an impact upon the manner in which social science computing is performed. Previous reliance upon batch computing systems forced social science programs into a mold in which output decisions had to be made simultaneously rather than sequentially, computations producing large amounts of (often unread) output were the rule, and errors in input preparation resulted in increased computer costs and turnaround time delays that were generally large relative to the magnitude of the error. In general, batch systems have had the effect of separating a non-programming investigator from his procedures and his data.

Recent uses of interactive systems provide convincing evidence that it is possible to exploit such hardware in a manner that allows a user considerably more freedom and flexibility with his data and the procedures he employs. The TROLL system[12] provides an example of such a use. Furthermore, in some general interactive systems including the first, CTSS,[4] batch and interactive modes of computation are not antithetical but combine naturally to use common system resources. The appropriate mode of computation may be chosen according to the characteristics of the task and the preferences of the user. Recent systems such as Digital Equipment Corporation's PDP-10 include interactive-batch compatibility as a basic design feature; such a feature is not costly relative to the benefits it provides and should be more common in the future.

The widespread availability of interactive systems plus the development of faster and less expensive user terminals should cause a shift of much social science computing to these systems. Among the reasons are: (1) the ability to specify tasks of analysis incrementally rather than collectively; (2) the potential of on-line interaction with data bases and the ability to browse through them with flexible computer support functions; (3) the generally lower cost of making a mistake in procedure and the ability to detect it more rapidly; and (4) the resynchronization of the user's processes of hypothesis formulation, testing, and analysis and the computer's use in supporting these functions. Batch processing will not and should not disappear; it will continue to be used for routine tasks in which interaction is not required or is inappropriate.

The computer terminal market has just begun a period of rapid development. In the past the terminal industry was dominated by two large firms, AT&T and IBM, and the market for terminals grew slowly; the rate of innovation in the industry was sluggish. As interactive computing began to increase, the market for terminals grew rapidly and many new firms entered the industry. The results have been increased competition, more product variety for consumers, lower costs, and an increase in the rate of innovation. If the growth in interactive computing continues, the terminal market should continue growing rapidly. Bigger markets should lead to increased specialization, some of which will benefit social science computing directly. For example, a low cost interactive graphics terminal having moderate power is well within the scope of present technology. Only a larger market is required for it to be introduced commercially. Cheaper, faster and more powerful terminals will make widespread use of terminals economically feasible and will allow enchancement of the quality and power of interaction.

The development and availability of general purpose computer networks is important because they will remove significant limitations on market size for social science computing products. By a general purpose network I refer to a network that can transmit digital data between any two of its members without knowing the content of the transmission. Such a network might exist on a permanent basis, such as the ARPA network[18] with its dedicated interface message processors and reserved communications circuits, or it might consist only of interface protocols between its members with common carrier transmission initiated whenever any member wants to activate a part of the network.

Social science computing product markets are now segmented by locality and by machine type. One of the major reasons for this segmentation is the high cost of importing both programs and large files containing behavioral data. One of the major effects of such segmentation is that investment in each market is limited by its size. Thus, investments in special products are made and replicated in many small markets.

The existence of general purpose networks should alter this resource allocation pattern. Suppliers of programs and data bases will have access to a much larger market for their products; the prospect of larger returns should serve to increase both the degree of specialization of those producers and the amount of investment they are willing to make in their products.

For consumers of social science computing products, such networks will increase substantially the number and variety of products available to them. As an example, consider the problem of accessing data files residing at a remote computer center. In the past, data have generally been exported by moving the data file from its initial computing environment to the data requestor's computing environment; usually the environments are technically and administratively dissimilar. The technical problems involved in such transfers are solvable, but they are generally time consuming and tedious. Computer networks offer viable technical alternatives to this procedure. Using a network, it would be possible to transmit a request for processing and output to the computer environment in which the data are stored and transmit the results back to the requestor. If the cost of transmission is low and there are few administrative barriers to access, every user can regard any program or data file on the network as being within his own facility and available to him at only moderately higher cost. Alternatively, the data could be transferred directly to the user's computer, since the existence of the network implies functional interfaces between each computer in the network and the network circuit itself.

The availability of computer networks should substantially reduce the inconvenience and costs of accessing remote computers, provided that the institutional and administrative problems of interorganizational computer access can be solved. This will create larger markets for social science computing tools and will lead producers to raise their level of investment in the products they offer. In addition, there should take place a trend toward specialization in the construction of tools for use, and collection and maintenance of specialized data bases at various research-oriented computer centers. An example of this process is provided by TROLL,[12] an on-line system for macroeconomic estimation and simulation. TROLL is available to its customers on-line using the AT&T telephone network. Partially as an outgrowth of the development of TROLL, a Computer Center for Economics and Management Sciences has been established by the National Bureau of Economic Research to continue and extend the production of quantitative methodology and computing tools. It is anticipated that access to programs and data will be made on-line, using the telephone system to support network communication.

Initially, the existence of network communication links will be far more important than how the network is implemented. The existence of some form of interprocessor communication will allow substantial changes in how and where social science computing is accomplished. The configuration and the nature of implementation of the network is a matter of economics; it depends upon such factors as anticipated pattern and volume of traffic, number and location of subscribers, and required response characteristics. Until a general purpose network becomes available, more primitive network communication can be supported using the existing telephone network. It seems unwise to build a computing network dedicated to social science computing traffic either now or in the near future. Such an act would be comparable to allocating some of the industry's limited resources to design special computer architecture that would be dedicated to social science computing.

Of all components of computer hardware, the current status and use of data storage technology represents a major constraint on the fashioning of better social science computing programs. As social science programs grow in scope, they grow in memory required also. It is now common for large social science applications systems to use elaborate manual segmentation systems to accommodate themselves to a small amount of address space. Either the memory resources for ameliorating these restrictions do not exist, or the techniques for accessing them are not known or available.

In the absence of a fundamental breakthrough in providing substantially lower cost immediate access memory, computers with virtual memory organization[5] appear to offer a solution to this problem. Virtual memory machines provide both increased program address space and data address space; the latter would provide the ability to process more complex data structures without making a substantial investment in file manipulation software. Virtual memory environments for building and using social science computing tools should obviate some of the ungraceful degradation characteristics which many social science computing programs exhibit upon reaching a capacity constraint. Perhaps most important, the availability of virtual memory would remove most memory management concerns from programmers and allow them to concentrate upon producing more useful computing tools. The opportunity cost of *not* using automatic memory management techniques is measured in terms of programmer time used in its place and investigator time waiting longer for results. This cost is already substantial in some cases, and it will increase as the cost of hardware declines relative to the cost of labor.

If the technical developments described above materialize, then their impact upon the social science computing industry should have very beneficial effects upon its performance. However, several problems exist within the industry that detract from its performance and that are not likely to be altered by changes in technology.

The first problem concerns access to data. While there has been much progress in recent years by private firms making data available in machine readable form, most behavioral and economic data generated by government agencies and private researchers is unknown to other potential users or underutilized by them. The principal reasons appear to be that: (1) within government agencies, quasi-monopoly positions have discouraged efficiency in both production and distribution; and (2) in academic and research institutions, the prevailing reward structure offers few professional rewards for data production alone. As a result, it is common to observe much duplication of data collection activity, underutilization of existing data sources, and lack of availability of existing timely and detailed data for research and policy analysis.

Another problem is closely related. Social science disciplines currently assign lower status to most "technical" work and higher status to "substantive" work. To the extent that this status difference is perceived, new entrants into social science professions are more likely to elect to do substantive work. Those entrants with strong preferences for the computational aspects of the social sciences are more likely to gravitate toward academic computer science rather than toward involvement in a service role in which room for professional growth appears limited.

Within the present industry structure, academically oriented producers of both programs and data find it difficult to obtain academic rewards for their output. The production of a useful, accurate and well-documented social science computer program or machine readable data file may generate substantial externalities for social science as a whole, yet there is no adequate mechanism that allows its producer to capture these externalities. The long run effect of this situation must be to discourage production by those able to produce successfully in other fields.

The problem of developing a reward structure for the social science computing industry is not a new one. Its solution could result in substantial benefits to social scientists in terms of substantially more efficient use of the computing and data resources available to them.

CONCLUSION

During the next ten years there will be a substantial increase in demand for social science computing products from government, universities, research institutions and private businesses. A continued decline in the cost of computing will benefit all these consumers. In addition, the development of reliable general purpose interactive systems and inexpensive terminals will enhance the quality and power of social science computing activities; general purpose networks will enlarge traditional markets and lead to increased competition and better consumer choice; and the development of quite large addressable memories should remove an important existing constraint on the development of large applications systems and complex file processing tools. These developments should be supported by the social science computing industry. Problems concerning access to data and achieving a reward structure for social science computing exist and are important, but remain to be solved in a satisfactory manner.

ACKNOWLEDGMENTS

I would like to acknowledge helpful discussions with and criticisms from Mr. Ben A. Okner and Mr. Gregory A. Marks.

REFERENCES

1 J W BEAN S W KIDD G SADOWSKY
B D SHARP
The BEAST: A user oriented procedural language for social science research
Processed; Washington The Brookings Institution June 13 1968

2 R BUHLER
P-STAT: An evolving user-oriented language for statistical analysis of social science data
(Describes P-STAT tape version 42.) Processed; Princeton Computer Center Princeton University 1966

3 A S COOPERBAND W H MOORE JR
R J MEEKER G H SHURE
TRACE-III: An implicit programming system for inductive data analysis
Proceedings of the 1971 Annual Conference of the Association for Computing Machinery New York 1971 pp 127-138

4 P A CRISMAN ed
The compatible time-sharing system: A programmer's guide
2nd ed Cambridge The MIT Press 1965

5 P J DENNING
Virtual memory
Computing Surveys Vol 2 September 1970 pp 153-189

6 W J DIXON ed
BMD biomedical computer programs
Berkeley and Los Angeles University of California Press 1967

7 M GREENBERGER M M JONES
J H MORRIS JR D N NESS
On-line computation and simulation: The OPS-3 system
Cambridge The MIT Press 1965

8 *The data-text system: A computer language for social science research designed for numerical analysis of data and content analysis of text*
Harvard University Department of Social Relations (Preliminary Manual) Processed; Cambridge Harvard University 1967

9 J M HENDERSON R E QUANDT
Microeconomic theory: A mathematical approach
New York McGraw-Hill Book Company 1958

10 C C HOLT
A system of information centers for research and decision making
American Economic Review Vol 60 May 1970 pp 149-165

11 J W LONGLEY
An appraisal of least squares programs for the electronic computer from the point of view of the user
Journal of the American Statistical Association Vol 62 September 1967 pp 819-841

12 W MALING
Preliminary version of the TROLL/1 reference manual
Processed; Cambridge Massachusetts Institute of Technology 1971

13 M C McCRACKEN
A computer system for econometric research
Processed; Ottawa Economic Council of Canada 1966

14 S D McINTOSH D M GRIFFEL
The ADMINS primer—November 1968
Processed; Cambridge Massachusetts Institute of Technology (Center for International Studies) 1968

15 E D MEYERS JR
Project IMPRESS: Time-sharing in the social sciences
AFIPS Conference Proceedings 1969 Spring Joint Computer Conference Montvale New Jersey AFIPS Press Vol 34 1969 pp 673-680

16 N NIE D BENT C H HULL
SPSS: Statistical package for the social sciences
Hightstown New Jersey McGraw-Hill Book Company 1970

17 F J REX JR
Herman Hollerith, the first statistical engineer
Computers and Automation Vol 10 August 1961 pp 10-13

18 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
AFIPS Conference Proceedings 1970 Spring Joint Computer Conference Montvale New Jersey AFIPS Press Vol 36 1970 pp 543-549

19 S ROSEN
Electronic computers: A historical survey
Computing Surveys Vol 1 March 1969 pp 7-36

20 D L ROSS
Burroughs Advanced Statistical Inquiry System (BASIS)
Processed; Detroit Burroughs Corporation 1969

21 R H WAMPLER
An evaluation of linear least squares computer programs
Journal of Research of the National Bureau of Standards Vol 73B April-June 1969 pp 59-90

A computer model of simple forms of learning in infants

by THOMAS L. JONES

National Institutes of Health
Bethesda, Maryland

INTRODUCTION AND SUMMARY

Many workers have studied the problem of getting machines to exhibit aspects of intelligent behavior. It has become clear that a major limitation of the artificial intelligence field is the cost and difficulty of programming, which remains essentially a handicraft technique. What we need in artificial intelligence research are methods for the machine to be self-programming in a much deeper sense than the compilation process in use today. Briefly, we would like for the machine to "learn" to solve a problem, rather than being programmed in the usual laborious way. This report presents a computer program, using what I will call an "experience-driven compiler," which is able to program itself to solve a restricted class of problems involving cause-and-effect relationships. Although the program is not yet able to learn to solve problems of practical significance, it is hoped that the technique will be developed to the point where this can be done. I am in agreement with Turing's suggestion (1950) that the best way to achieve artificial intelligence is to find out what an infant has in his brain that allows him to become intelligent, put this capability into a computer, then allow the machine to "grow up" in much the same way as the baby does. (Of course, this is a statement of the problem rather than an explanation of how to solve it.) An additional purpose of the work is in psychology: we would like to have computer methods for testing theories of how learning might occur in living creatures.

There have been many reports of techniques for getting machines to exhibit various forms of learning (Friedberg, 1958; Samuel, 1959; Waterman, 1970). Usually they are concerned either with coefficient learning, where some parameter given by the human programmer is optimized, or with rote learning, involving the formation of some table or file of data. This report describes a different form of learning, *program learning*, where the output of the learning process is a computer program. Greene and Ruggles (1963) described a computer model of behavior in human infants. In psychology, Piaget (1952) and Hull (1952), among others, have studied the process of behavior acquisition. The methodology used in the present report can be considered an extension of Hull's method of formulating assumptions in a precise, formal way, deducing the consequences of the assumptions, and comparing the results with observed behavior. The computer permits us to evaluate a learning model by turning it on and running it; the model's capabilities are there for all to see, and its weaknesses are glaringly obvious. The model may be made more like the real organism by improving it in an engineering sense, since a dissimilarity is almost always something which the living organism can do and the model cannot.

This report describes a computer program called INSIM (for "INfant SIMulator"), written in LISP (McCarthy, 1960) for the PDP-10 computer. The program operates by learning cause and effect relationships, such as "turn head" causes "mouth touch." A cause-effect chain is formed; if A causes B, and B causes C, and if C is pleasurable, the program will work backward along the chain until it reaches something it can control directly (a muscle pull). The simulated infant's behavior is under the control of a so-called performance program. The performance program is, in turn, written, as learning proceeds, by a section of the learning program called the "experience-driven compiler."

A typical problem solved by INSIM involves a behavior pattern which is all too familiar to parents of small children: That of sucking the thumb. (We will soon see that this involves quite a virtuoso feat of information processing.) The problem can be described in logical notation as follows:

(1) object touching mouth→pleasure
(2) (left cheek touch AND turn head left)→
mouth touch

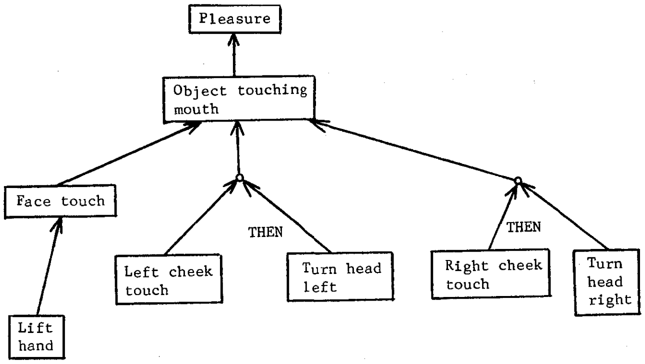


Figure 1

(3) (right cheek touch AND turn head right)→ mouth touch
(4) (left cheek touch OR right cheek touch) = face touch
(5) face touch→mouth touch (sometimes)
(6) lift hand→face touch

(See Figure 1.)

After the program has learned these connections, it will emit the behavior sequence, "lift hand, turn head (left or right)," resulting in pleasure.

Figure 2 is a block diagram of INSIM:

The *performance program* has the direct responsibility for synthesizing behavior. It is written in an interpretive language called PSIM (parallel simulator). The performance program receives stimuli from and sends responses to a *body and environment simulator*. The *display section* provides monitoring on the cathode-ray tube. The *motivation section* activates the main goal (oral gratification or curiosity).

Relatively little of the performance program is innate. Most of it is generated by an *experience-driven compiler* which is the core of the learning part of the program.

Causality is detected by statistical correlation; if a signal occurs on line A followed by one on line B, and if this sequence is repeated sufficiently many times, the program assumes that A causes B. The program is equipped for the simplest type of pattern recognition and concept formation: the formation of logical AND's and OR's of previously known variables. The program has an intellectual motivation system which causes it to exhibit simple forms of curiosity, play, and exploratory behavior.

THE PERFORMANCE PROGRAM

As described above, the performance program has the direct responsibility for receiving cues from the environment and emitting properly timed and sequenced behavior. The performance program consists of a set of PSIM statements arranged in a *goal tree*, each node of which is a *variable cluster*. A variable cluster consists of a *basis variable* such as "mouth touch," together with variables which indicate the status of the goal defined by the basis variable (e.g., achieving mouth touch). These include:

(1) The PR of the goal. This is defined as the probability that the goal can be achieved "soon" (in a sense to be defined precisely below), provided that an attempt is made by the infant to achieve it.
(2) The C (cost) of the goal. This is the program's estimate of the time delay required to achieve the goal.
(3) The WANT of the goal. This is TRUE or FALSE depending on whether or not the program is trying to achieve the goal. A goal is defined to be *active* if its WANT variable is TRUE: i.e., if the program is trying to achieve it.

In addition, the variable cluster associated with a goal contains other variables as described below.

The performance program, then, consists of a set of variable clusters for various goals, with *connections* between variable clusters for goals which are causally related. Thus, there will be a connection between the variable clusters for "turn head left" and "mouth touch." Figure 3 shows the goal tree for the thumb-sucking problem, with the variable clusters shown explicitly.

Psychologists will wish to compare this setup with Guthrie's "connectionism" concept (1952). Note again the resemblance to a nerve net. INSIM may be interpreted as a neurological model of how learning might take place in the brain, although, of course, it is a very

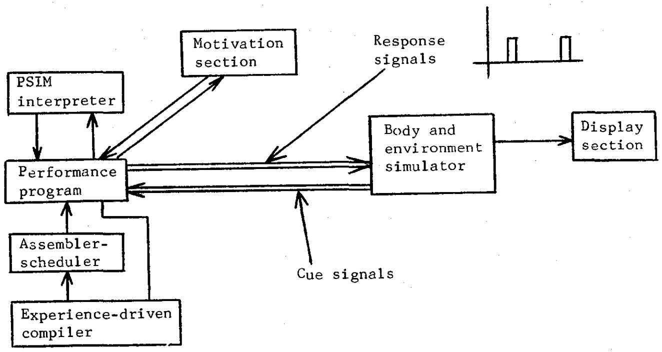


Figure 2

imperfect model and one for which no direct neurological evidence exists.

The performance program operates by *activating* various branches of the goal tree at the appropriate times. Recall that the active nodes of the tree are the nodes which define goals which the machine is trying to achieve at the current time. In the thumb-sucking problem, assume that the motivation section has activated the main goal "oral gratification." The performance program will activate the extreme left branch of the goal tree (see Figure 1), ending in "lift hand." The "lift hand" response will be emitted to the body and environment simulator. After a delay of roughly two simulated-time seconds, a cue, e.g., "left cheek touch," comes back, indicating that the simulated hand has been lifted to touch the (simulated) left cheek. Next, the branch ending in "turn head left" is activated. A "mouth touch" signal comes back from the body and environment simulator, indicating that this goal has been reached, and the motivation section turns on the oral gratificaticn flag, "rewarding" the program for its successful effort.

The basic problem is to decide which branch of the goal tree to activate. (INSIM performance programs allow only one branch to be active at a time; hence there is no way to work on two goals simultaneously.) In a given situation, the decision is made in two phases, a *feasibility study phase* and a *choice phase.* In the feasibility study phase, each branch of the tree is assessed, and an estimate is made of which branch is the quickest and surest way to the main goal. In the choice phase, this branch is activated. During the feasibility study phase, the PR (probability) and C (cost) are computed for each goal.

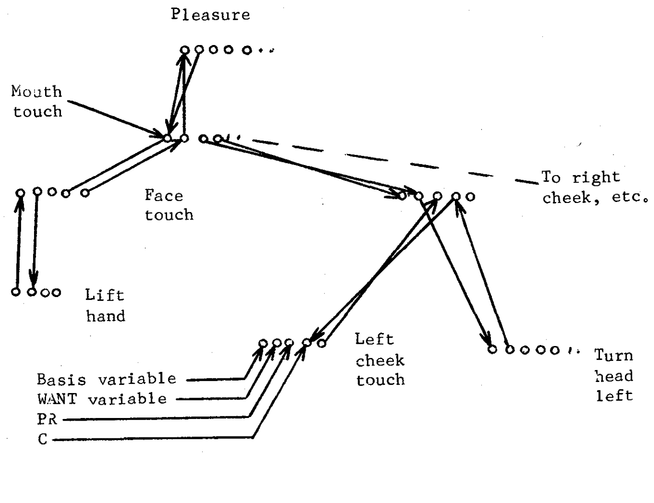


Figure 3

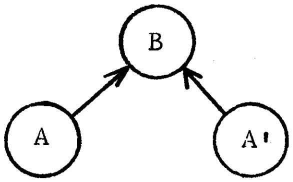


Figure 4

Computation of PR and C

This section is devoted to a detailed discussion of how PR and C are computed. On a first reading, readers may skip to the section on the choice phase.

PR is defined recursively as follows:

(1) For a "response" (directly controllable) variable, such as "lift hand" or "turn head left," PR = 1.

(2) Suppose that A is one of several OR'ed subgoals of B (see Figure 4).

If A is the "best" subgoal according to a criterion to be presented momentarily, then PR(B) = PR(A) Pr(B | A), where Pr(B | A) is the conditional probability of B given A (i.e., the probability of getting from A to B) as estimated by the coefficient learning program PRDLRN, discussed below.

The "best" subgoal is selected so as to maximize the Slagle coefficient (1964)

$$\frac{\mathrm{PR(B)}}{\mathrm{C(B)}}$$

This subgoal may not really prove to be the best one, if, say, the probability or cost estimates turn out to be incorrect. The performance program is a heuristic program and is forced to make decisions based on imperfect evidence.

A more sophisticated program would take into account the possibility of trying to achieve A, failing, then trying to achieve A′ and succeeding. Thus a goal with several good subgoals would have a larger PR for this reason. However, INSIM considers only the one best subgoal.

(3) Suppose that A1 and A2 are components of the ordered-AND goal A1THA2 (A1 then A2). Then PR (A1THA2) = PR (A1) PR (A2).

(4) Notwithstanding any of the above, if a goal has already been achieved, its PR = 1. A goal is defined as "already achieved" if the corresponding signal has occurred within the last five seconds.

Similarly, the C (cost) of a subgoal is defined recursively as follows:

(1) For a response variable, C = 0.
(2) If A is the best of several OR'ed subgoals of B, then C (B) = C (A) + PR (A) Delay (A→B).
(3) The C of an ordered-AND goal A1THA2 (A1 then A2) is C (A1THA2) = C (A1) + PR (A1) C (A2)
(4) Notwithstanding any of the above, the C of a goal is 0 if the goal is already achieved (in the past five seconds).

To summarize, in the feasibility study phase, estimates are made of the success probability and time delay of each path to the main goal.

The choice phase

The next step is to activate the goal tree branch which is estimated, according to simple heuristics, to be the quickest and surest path to the main goal. A goal is *active* if, and only if, its WANT variable has the value TRUE.

(On a first reading, readers may skip to the section on the inner loop.) The WANT variable of a goal A is defined recursively as follows:

(1) If A is a main goal, WANT (A) is TRUE or FALSE as set by the motivation system.
(2) If A is one of several OR'ed subgoals of B, WANT (A) = (WANT (B)) AND (A is not already achieved) AND (A is the best subgoal of B)) OR
  (A is a curiosity goal (see below)).
(3) If A1THA2 is the ordered-AND subgoal "A1 then A2,"
WANT (A1) = WANT (A1THA2) AND (A1 is not already achieved)
WANT (A2) = WANT (A1THA2) AND (A1 is achieved) AND (A2 is not achieved).
(4) If G is a response (directly controllable) variable, WANT (G) causes the response to be emitted.

The inner loop

Since the computation of the PR, C, and WANT variables is under control of PSIM, they will be recomputed in a wavefront-like manner, with the stimulus variable as the starting point of the wave. Thus the PR and C and the program's decisions are constantly being updated on the basis of changing conditions. PSIM will update only those tree branches which depend on some variable which has changed since the last TCLOCK time.

Discussion

INSIM performance programs incorporate simple heuristics which work well in cases where the assumptions on which they are based hold true.

Among the assumptions are:

(1) Success probabilities and time delays are assumed to be statistically independent. If this is not true, the chaining formulas used in computing success probabilities and time delays will not be accurate.
(2) It is assumed that goals do not conflict; i.e., that the achievement of one goal does not decrease the probability of achieving another goal.

Removing these performance limitations would require additional machinery beyond the scope of the INSIM project, such as a look-ahead method of the type used in chess programs.

THE EXPERIENCE-DRIVEN COMPILER

As mentioned previously, most of the performance program is coded by an internal compiler which, instead of using as its input a source code prepared by a human, is controlled by the experience acquired by the program as it interacts with its (simulated) environment. In keeping with the dictum that in order to learn something, one must know something already, the compiler incorporates the probability and delay formulas described above, plus knowledge of basic aspects of the physical world, including time and causality (see Figure 5).

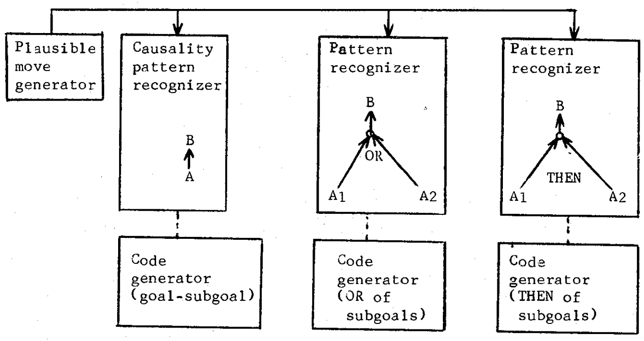

Plausible move generator
Causality pattern recognizer
B
A
Pattern recognizer
B
OR
A1
A2
Pattern recognizer
B
THEN
A1
A2
Code generator (goal-subgoal)
Code generator (OR of subgoals)
Code generator (THEN of subgoals)

Figure 5

The plausible move generator is used instead of testing for causality between all possible variables A, B. The latter approach would involve on the order of n^2 tests, where n is the number of variables.

It is the compiler which sets the upper limit on the program's ability to learn. For example, INSIM could never learn to play chess even with very long training, because the necessary pattern recognizers and code generators are not present.

The experience-driven compiler operates as follows: The program starts out with an innate main goal which is "oral gratification" in the thumb-sucking problem. First, the plausible move generator is called to generate a list of variables which are likely to be "relevant" to the oral gratification goal, and *causality test links* are formed. (The plausible move generator is discussed below.)

Next, the *causality pattern recognizer* learns which test links represent actual causal relationships. The pattern it is looking for is shown in Figure 6.

If a pulse on variable A is followed by a pulse on variable B sufficiently often, A is assumed to cause B. More precisely, if $\Pr(B \mid A) > 0.20$ after at least 15 pulses on A and 15 pulses on B have occurred, A is assumed to cause B. The pulses on A and B must be less than five simulated-time seconds apart. (If there are any pulses at all on B, then a pulse on A will always be "followed" by a pulse on B if we wait sufficiently long.) $\Pr(B \mid A)$ is estimated by the *probability-delay learner,* discussed below.

The simple heuristics will miss some actual causal relationships when the delay is more than five seconds. It would not be hard to make the program "adaptive" to this arbitrary parameter, say by matching the time delay to the recent density of pulses on the two lines.

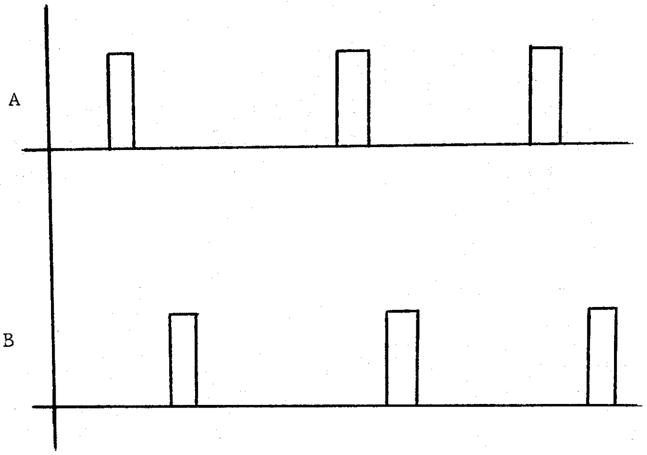


Figure 6

Thus, if there were only one pulse on B about every 15 minutes, the allowable delay might be five minutes rather than five seconds. Also, the heuristics will sometimes "identify" a causal relationship where none actually exists. E.g., if the allowable time delay were long enough, the program would think that day causes night. (Piaget has found that small children also think that day causes night.)

In some cases, it is sufficient to wait passively for a pulse on A. In other cases, the *curiosity section* of the performance program sets WANT (A) to TRUE, activating some goal tree branch ending in A and initiating behavior which hopefully will lead to a pulse on A, in order to see if B follows (e.g., it activates "turn head left" to see if "mouth touch" follows): this is the "play" or "exploratory behavior" mentioned above. The curiosity section attempts to test links which are new and have not been tested many times; links where the initial variable, A, is reasonably easy to obtain; and where the final variable, B, is "biologically useful" in that ability to obtain B would contribute to the program's ability to obtain pleasure (primary reward). Specifically, the curiosity section tests the link A→B which maximizes

$$(PR(A)/C(A)) * Satfunc(A, B) * Need (B)$$

where Satfunc (A, B) (saturation function) decreases linearly from 1 to 0 as the number of times when A→B has been tested increases from 0 to 15. Need (B) is an index of how much the ability of the program to obtain primary reward would be improved by improvements in its ability to obtain B. See Jones (1970) for a description of the heuristics used to compute Need (B). Only links which have been designated as "plausible" by the plausible move generator are tested.

When the causality pattern recognizer detects that two variables, A and B, are causally related, the corresponding code generator is called to compile the link A→B in the goal tree, provided that the connection has not already been made. This code generator is a LISP function called MAKEORGOAL (A, B), so named because it also handles the case where A is one of several logically OR'ed goals. In LISP, the code generator turns out to be a straightforward and rather prosaic, if slightly long, program. Separate sections are provided for compiling the entries for WANT, PR, C, and each variable associated with the curiosity system. Each section looks up the names of the variables involved in the formula in question and substitutes them in the formula, using LISP's symbol-substituting capability.

In the thumb-sucking problem, the program first learns the links shown in Figure 7.

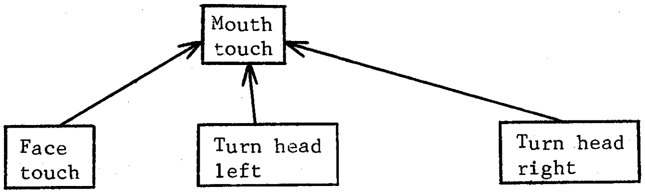


Figure 7

Although this version of the performance program will sometimes succeed in obtaining "mouth touch," it does not yet know which way to turn the (simulated) head. Next, the plausible move generator is called to provide a list of variables to be THEN'ed with the partially successful subgoals. Causality test links are compiled for the ordered-AND variables. Among them are the ones shown in Figure 8.

V1 correlates very poorly with mouth touch; V2 correlates very well. Since Pr (mouth touch V2) is very high, the performance program will activate this branch, rather than the others, and the simulated infant will emit "turn head left" in response to "left cheek touch." Similarly, it learns to emit "turn head right" in response to "right cheek touch." Finally, "face touch" is identified as a "biologically useful" variable, and the program learns to activate "lift hand"; when the (simulated) hand touches the face, the previously learned program takes over and completes the thumb-sucking operation.

One way of looking at the learning process is that the program builds a subroutine hierarchy. Each node on the goal tree defines a subroutine, the process of achieving a stimulus on the variable defined by the node; e.g., the "obtain mouth touch" subroutine. Each link on the tree defines a subroutine call. Thus, the "obtain face touch" subroutine calls the "lift hand" subroutine.

See Jones (1970) for a fuller discussion of how the experience-driven compiler is organized and programmed.

It is interesting to note the similarity between the learning sequence and Piaget's observations on the learning of human infants. Although the real infant's learning is much more complicated, it follows the same gross sequence of stages; the real infant first learns to search from left to right with its head; then it learns which way to turn; then it learns to lift its hand and suck its thumb. (Real infants also show considerable variability in their learning; for example, some have been observed to suck the thumb in the womb.)

PSIM

In order to simplify the experience-driven compiler, an interpreter called PSIM (parallel simulator) was written. The experience-driven compiler "sees" a pseudo-machine which is quite different from the actual PDP-10 for which the program is written. The pseudo-machine is much like an analog computer in which each component has the versatility of a digital computer, but where one does not need to worry about the sequence of computations; instead, each component "continuously" monitors its input lines and responds to whatever signals it finds there. Thus PSIM is a "stimulus-response" oriented interpreter which is convenient for writing programs which simulate actions in the real world.

More precisely, a PSIM program consists of a network of *variables* whose values change with simulated time and *functional relationships* which describe the way the variables depend upon each other. There is a *simulated-time clock*, TCLOCK, calibrated in seconds, which is set to zero at the start of a simulation and advances as the simulation proceeds. There is an *event file* which contains computations which are scheduled to occur at a future TCLOCK time. Each PSIM entry is of the form VAR = EXPR, where the EXPR is a LISP expression which is evaluated to get the value of the variable VAR.

If a variable V2 depends on another variable V1, and V1 changes, V2 is automatically updated. This process occurs in an interesting way: In addition to the variable TCLOCK, which records simulated time, there is another "clock," called BTIME (base time) which cycles from one up to some maximum value during each TCLOCK time. Each variable has a base time at which it is updated. The base time of a variable is set as follows:

(1) If a variable depends on the other variables only through a TCLOCK delay, its base time is 1.

(2) Otherwise, its base time equals one plus the maximum of the base times of the variables on which it directly depends; i.e., the variables in its EXPR.

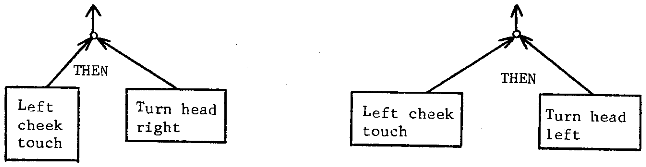


Figure 8

Variables with base time 1 are updated first, then variables with base time 2, etc. This arrangement ensures that a variable is not updated until after the correct values of all variables in its EXPR are available.

During most TCLOCK events, only a few variables change their values. PSIM ensures reasonable efficiency by recomputing only those variables which depend on a variable which has changed at the current TCLOCK time. The updating proceeds in a "wave-front like" manner, starting with a variable X which has been changed through a TCLOCK delay, then updating the variables V_i which depend on X, then the variables which depend on some V_i, etc. (See Figure 9).

Note the resemblance to a hard-wired device or a nerve net.

THE PROBABILITY-DELAY LEARNER

Conditional probabilities and time delays are estimated by a rather orthodox coefficient learning procedure (Samuel, 1959). Suppose there is a link between A and B. Whenever A occurs, followed within five seconds by B, Pr (B | A) is incremented by an amount θ (1-old value of Pr (B | A)), and Delay (A→B) is incremented by θ (actual delay—old estimate of delay). If A occurs, but not B, Pr (B | A) is decremented by an amount θ (old value of Pr (B | A)) and Delay (A→B) is incremented by θ (five seconds—old estimate of delay). It can be shown that this procedure gives unbiased estimates of Pr (B | A) and Delay (A→B), with an exponential weighting such that old occurrences of A affect the estimates less than new ones. θ, the decay coefficient, is currently 0.1. The initial estimate of Pr (B | A) is obtained by observing the first 10 occurrences of A. Pr (B | A) is set to

(Number of A's followed by B's)/(Number of A's)

The initial estimate of Delay (A→B) is the observed average over the first 10 occurrences of A.

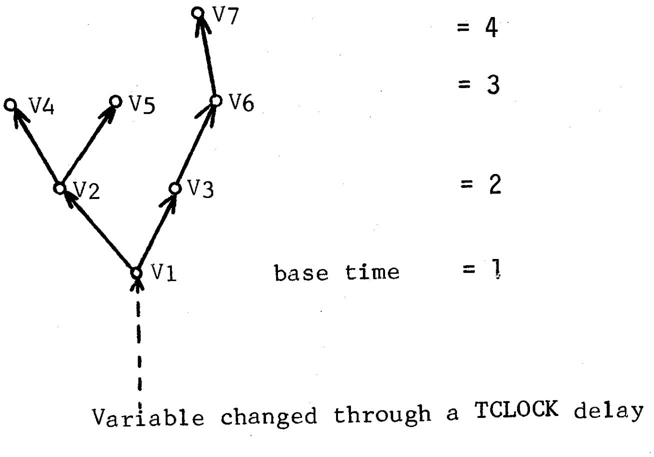


Figure 9

$$\begin{array}{ccccc}
V_1-V_1 & V_1-V_2 & V_1-V_3 & V_1-V_4 & \cdots \\
V_2-V_1 & V_2-V_2 & V_2-V_3 & V_2-V_4 & \cdots \\
V_3-V_1 & V_3-V_2 & V_3-V_3 & V_3-V_4 & \cdots \\
V_4-V_1 & V_4-V_2 & V_4-V_3 & V_4-V_4 & \cdots \\
\vdots & \vdots & \vdots & \vdots &
\end{array}$$

Figure 10

THE PLAUSIBLE MOVE GENERATOR

Recall that, mention was made of a not yet implemented plausible move generator which was to be called to make hypotheses about which variables might be causally related to some goal B. These variables are defined as "relevant" to B; the causality pattern recognizer will determine if an actual causal relationship exists. (At present, the program is running with a patch which merely looks up plausible variables in a manually prepared table.) If we tested all possible variable pairs, the machine time needed would increase on the order of n^2 where n is the number of variables, constituting an intolerably large CPU and core storage load for large n. The plausible move generator will use three relevance heuristics:

(1) (implemented) Innately known relevance—A variable A is innately known to be relevant to a variable B if A appears under B in an innately known file (i.e., a manually prepared file).

(2) The diagonal search—This will be used to associate variables which are very "important," such as a "large moving visual stimulus" and "arm motion," and will be used to make the initial connections between sensory modalities (e.g., vision and hearing) and motor units such as the arms or the head. The "important" variables will be placed on an innate list in decreasing order of importance. Let V_k be the kth most important variable. Make a square matrix as shown

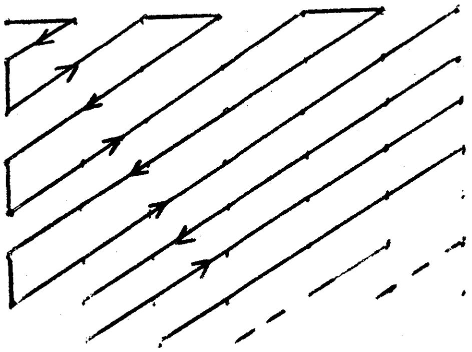


Figure 11

in Figure 10. Now let a dot represent a matrix entry and search the matrix in the order specified by the arrows in Figure 11. Roughly, the concept is that the most important variables are associated with each other first.

(3) The relevance chainer and the innate net—The innately known relevance subroutine and diagonal search subroutine will be used to get the learning process started. To continue it, the *relevance chainer* section of the plausible move generator will be used. The relevance chainer will incorporate two heuristics: (1) If x is relevant to y, then y is relevant to x (symmetry). (2) If x is relevant to y, and y is relevant to z, then x is relevant to z (transitivity). A depth cutoff will be used, so that all variables are not relevant to each other.

The relevance chainer will be used with an *innate net* which will incorporate innate knowledge about which sensory modality a given signal belongs to and innate information about space (which signals "belong together"). For a simple example, assume a one-dimensional space of touch receptors. The receptors will be arranged in an OR tree with a hierarchy of larger and larger sectors (see Figure 12). (Do not confuse this with a goal tree; the upper level variables are defined as OR's of the lower level variables.)

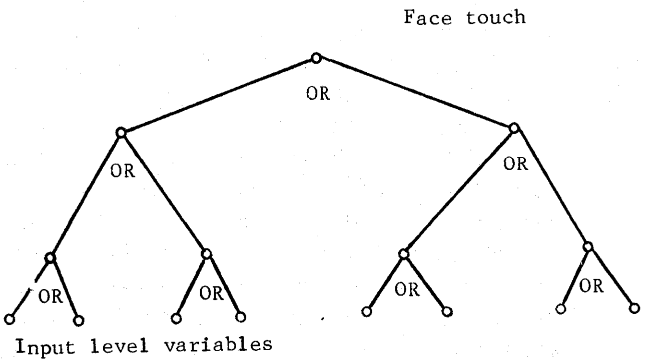


Figure 12

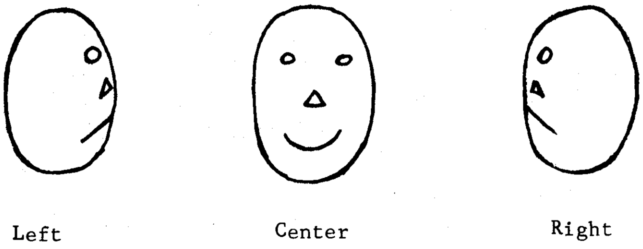


Figure 13

EXPERIMENTAL RESULTS

INSIM was tested with a simple body-and-environment simulator, with a problem environment which is a simplified version of that seen by a newborn human infant. Let us follow the growth of the simulated infant as it develops, starting from a state of passivity, then learning gradually to distinguish right from left and turn its head in the proper direction. Its ultimate achievement is to suck its thumb.

The simulated infant has a head with three positions, left, center, and right (see Figure 13). It has a blanket (Figure 14) and a bottle (Figure 15). The blanket and bottle are under the control of special manually prepared subroutines in the test package.

The simulated infant begins its life with an essentially blank performance program. At first it is completely helpless and does not move at all, since there are no motor operators in the goal tree. When it gets hungry, or, more precisely, wants pleasure, it must wait passively until its bottle appears. Although it is immobile, the infant is learning the connection "mouth touch" yields "pleasure."

Figure 14

At age 605 seconds, the experience-driven compiler is activated. The new performance program contains additional lines of PSIM code which express the goal tree link "mouth touch" yields "pleasure." The new PSIM code is then fed to the assembler-scheduler, which makes up, for each variable V, a list of variables which depend on V and a list of variables which V depends upon. Next, the scheduler assigns a "btime" (base time) to each variable, starting with btime = 1 for input variables such as mouth touch. The new performance program contains the connection "mouth touch" yields "pleasure." Mouth touch has become "important" to the infant; hence the new performance program is "curious" about things which are plausibly related to mouth touch. Among these are the head-turning responses. There are two head-turning operators, "turn head left" and "turn head right." After 10 seconds, the head automatically returns to the center position. This version of the performance program turns the head only out of curiosity; when it actually activates the mouth touch goal, it is unable to turn the head. Meanwhile, the connection "face touch" yields "mouth touch" has been made, and the "lift hand" response is enabled. Whenever the hand control subroutine receives a "lift hand" response, it computes a pseudo-random number and uses it to determine the outcome of the lift-hand operation; the hand winds up at either the left, center, or right positions relative to the infant's face.

At age 1405 seconds, the head-turning operators become subgoals of mouth touch. In this stage, the program exhibits for the first time a simple form of what psychologists call "operant conditioning" (Skinner, 1953). In the past, the program has been rewarded by a mouth touch for turning its head. Now, when it wants mouth touch, it turns its head. The program is still not very successful. It exhibits a stereotyped behavior pattern, first turning right, then left, without regard to where the object is, or, indeed, whether there is an object present or not. During this phase, the program is keeping records of the conditions under which "turn head left" and "turn head right" succeed in achieving mouth touch, laying the groundwork for the next phase of development.

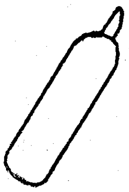


Figure 15

At age 2405 seconds, the sixth and final version of the performance program is compiled. When an object is present, the program can reliably turn the head in the proper direction. Thus the program exhibits a simple form of what psychologists call "discrimination learning"; it responds appropriately to the stimuli "left cheek touch" and "right cheek touch." The program's most advanced behavior involves providing its own stimulus object: its hand. This exhibits what psychologists call "chaining." Let us review the operating principles of the program by tracing in some detail what happens when, say, a "left cheek touch" pulse is received from the body-and-environment simulator. Variables which depend on "left cheek touch" are recomputed in a "wave-like" fashion, starting with variables which depend directly on "left cheek touch," then variables which depend on the latter class of variables, etc. Before the "left cheek touch" signal is received, the program assigns a low PR to achieving "left cheek touch." After the signal is received, the PR is set to one. Now the program assigns a high success probability to the composite goal "left cheek touch then turn head left"; previously, the composite goal also had a low success probability. This composite goal now has a higher success probability and figure of merit than the other subgoals of mouth touch; hence it is activated. Since the first half of the composite goal (left cheek touch) has already been achieved, the goal (and response) "turn head left" is activated.

ANALYSIS

The fundamental scientific issues to which the INSIM work addresses itself are:

(1) Can one make use of a relatively small amount of very general innate knowledge in order to obtain a much larger amount of specialized knowledge, and, if so, how?
(2) What should the innate knowledge be?
(3) How should the innate knowledge be incorporated into an information-processing system?

These issues are as old as epistemology itself, but the first really careful analyses were by Hume (1777) and Kant (1781). Hume took the position that the human mind was a "tabula rasa" (blank tablet) at birth and that all knowledge was acquired through the forming of associations (compare Hebb's [1949] synaptic connections). Kant, on the other hand, believed that the infant had a store of innate (categorical, or "a priori") knowledge at birth and that this was necessary to make the learning process work properly. The INSIM research supports Kant's viewpoint, based on practical engineering experience with information processing systems of this type. INSIM is associationist ("turn head left" is associated with "mouth touch"); however, in order to make the association proceed properly, innate knowledge had to be incorporated into the learning program, knowledge that there are such things as causality and time, and that causally related events are likely to occur in close temporal sequence.

A question immediately arises as to just what is meant by statements such as "INSIM knows that there is such a thing as causality." The word "know" can be used in several senses. Obviously INSIM does not know about causality in the same sense that an adult knows about causality; the program cannot explain causality, cite examples, or answer questions about causes and effects; it has no verbal behavior at all. Instead, to say that INSIM knows about a certain type of causality is to say that the INSIM program is optimized to a universe in which certain types of causal relationships exist. Thus, in a universe where there were no such things as causality or time, or where cause and effect were always separated by hours or days, INSIM would not work properly. In other words, the innate knowledge of INSIM is incorporated in the form of algorithms rather than as facts.

The learned knowledge is also incorporated into the system in the form of algorithms rather than facts. Thus the final version of the performance program knows right from left in the sense that it can turn its head in the proper direction if stimulated; yet it has no verbal knowledge of space at all. Expressing knowledge as algorithms, as in program learning, is meritorious in that it is algorithms which we know best how to combine into complex integrated systems. Thus INSIM learns a nursing subroutine, then adds additional code to form a thumb-sucking subroutine. By contrast, for a machine to learn facts is at present often like adding more books to a library; the machine cannot do much with them. Many of the theorem-proving efforts suffer from this problem. Expressing knowledge as facts has its complementary merits, as discussed by Hewitt (1969) and the present author (1966).

Given the concept of using a base of very general innate knowledge to obtain a much larger repertoire of learned knowledge, what can we say about what the innate knowledge (innate algorithm) should be; in particular, how much innate knowledge is needed and how problem-specific should it be. Why was the innate knowledge basis of INSIM chosen the way it was? For several reasons:

(1) The INSIM program has a high "bootstrapping leverage" ratio:

$$\frac{\text{(learned problem solving ability)}}{\text{(innate knowledge)}}$$

where the "amount" is to be indexed by some criterion or other. Although this is not a fact which can be easily demonstrated now, since the learning is severely limited by the body-and-environment simulator, INSIM is capable in principle of synthesizing very large trees of OR'ed and THEN'ed goals (limited by core storage), implementing long chains of behavior.

(2) INSIM is based upon a strong general theory of problem-solving, means-end analysis (Newell, Shaw, and Simon, 1959), and one can be confident of the program's ability to be extended to other goal types and to solve harder problems.

(3) INSIM performance programs are free to some extent from the "exponential explosion" problem which plagues many problem-solving systems. The simulated time to learn a tree grows only linearly with the length of the tree; the CPU time per simulated time second grows a little worse than linearly with the number of branches on the tree, and the number of branches is limited by the causality pattern recognizer.

(4) Lastly, the INSIM setup was chosen because it works; i.e., it constitutes an integrated learning-behaving system. This requirement is harder to meet than appears on the surface; I have 1100 pages of notes on setups which did not work, developed before arriving at INSIM. The incompletely successful efforts of Pavlov and Hull are also testimony to the difficulty of getting anything which will work at all.

Among the additional capabilities which are being considered for inclusion in INSIM are the ability to handle binary (non-pulse) goals, goals involving continuous valued variables, goal conflict situations, goals stated in terms of objects with property lists one-trial learning, and a look-ahead system of the type used in chess programs. An important lesson to be learned from a program of this type is that learning, rather than being a single process, actually involves a variety of ways for improving behavior based on experience.

A future objective of this work is to make it easier, in certain cases, for the machine to learn for itself how to do something than to write a program in the usual way. The learning program would need to be equipped for more advanced types of learning, including one-trial learning, and it would be restricted to problems involving long chains of cause and effect, such as robot control problems. In psychology, the objective is to use computer modeling, in conjunction with research on living creatures, to gain insight into that fascinating enigma—the human mind.

ACKNOWLEDGMENTS

This report is based in part on a Ph.D. thesis submitted to M.I.T. in September, 1970. I would like to thank Marvin Minsky and Seymour Papert for their support and encouragement, and for several ideas expressed in this paper.

REFERENCES

1 R M FRIEDBERG
A learning machine, Part I
IBM J Res and Dev Vol 2 pp 2–13 January 1958

2 P H GREENE T L RUGGLES
Child and Spock
IEEE Trans on Military Electronics Vol M17 No 2 and 3 April–July 1963

3 E R GUTHRIE
The psychology of learning
rev ed New York Harper 1952

4 D O HEBB
The organization of behavior
New York John Wiley and Sons Inc 1949

5 C HEWITT
PLANNER: A language for proving theorems in robots
Proc 1969 IJCAI pp 295–301

6 C L HULL
A behavior system
Yale University Press 1952

7 D HUME
Enquiry concerning human understanding
Oxford The Clarendon Press 1936

8 T L JONES
A computer model of simple forms of learning
MIT PhD thesis 1970

9 T L JONES
The advice-taker as a means of symbolic computation
MIT EE thesis 1966

10 I KANT
Critique of pure reason
Macmillan and Co Ltd 1934

11 J McCARTHY
Recursive functions of symbolic expressions
Comm ACM Vol 3 April 1960

12 A NEWELL J C SHAW H A SIMON
Report on a general problem-solving program
Proc Internatl Conf on Information Processing Paris UNESCO House 1959

13 J PIAGET
The origins of intelligence in children
New York International Universities Press 1952

14 A L SAMUEL
Some studies in machine learning using the game of checkers
IBM J Res and Dev Vol 3 pp 211–219 July 1959

15 B F SKINNER
Science and human behavior
New York Macmillan and Co 1953

16 J R SLAGLE
An efficient algorithm for finding certain minimum-cost procedures for making binary decisions
J ACM Vol 11 No 3 pp 253–254

17 A M TURING
Computing machinery and intelligence
in Computers and Thought pp 11–35 New York McGraw-Hill Book Company 1963

18 D A WATERMAN
Generalization learning techniques for automating the learning of heuristics
Artificial Intelligence Vol 1 Nos 1 and 2 Spring 1970

An information management system for scientific gaming in the social sciences

by ROBERT C. NOEL and THOMAS JACKSON

University of California
Santa Barbara, California

INTRODUCTION

During the past decade, the use of gaming techniques has spread among social scientists. Beginning primarily in the fields of international strategic studies and business management, gaming is now used in such areas as urban studies, general economics, political science, environmental studies, educational administration, and sociology. The purposes to which gaming techniques have been put are varied. Education and training applications have been by far the most numerous, and these continue to be the areas of broadest consensus about the value of gaming. Second most common are policy-analytic applications wherein present and future problems of policy choice are examined through gaming by expert analysts and policymakers. Finally, there have been basic research applications in which gaming exercises are used as settings for more or less controlled experimentation dealing with social psychological, sociological, and political variables.

The computer has played a role throughout most of the history of gaming. Computer support has usually taken the form of discrete, man-model simulation. That is, a computer-based model of some object system is solved reiteratively, taking the decisions of human participants as inputs and reporting changes in system states as stimuli for subsequent human decisions. Another kind of computer supported gaming differs from simple man-model simulation in that it involves two or more participating entities (teams) who, while interacting with each other directly in writing and/or face-to-face, also interact with each other indirectly through their interactions with a common computer-based model (for example, a multi-firm business game or an n-player international political game). Various names have been applied to such games, including "game-simulations."

Beyond performing these simulation functions, computer support for gaming has been far less common. Bloomfield has experimented with computer-based information retrieval as an aid to decision-makers in an all-man international strategic exercise, a so-called "free-form game," in which expert judgment performs functions analogous to the computer model in a man-model simulation (Beattie and Bloomfield, 1969).[1] Gerald Shure developed a system for team-to-team written communications on a time shared computer.[2] But, to our knowledge, no comprehensive computer-based system has yet been developed for the overall administration of gaming exercises.

The purpose of this paper is to describe such a system. It is under development at the POLIS Laboratory at the University of California, Santa Barbara.* Its first version is operational on a dedicated PDP-11. It is currently undergoing major revision; an expanded version should be completed within a year. Part 1 will give a non-technical description of the system, and Part 2 will provide technical information about the architecture of the system.

GENERAL DESCRIPTION OF THE POLIS SYSTEM FOR THE ADMINISTRATION OF GAMING EXERCISES

The system performs a variety of information management functions for gaming. First, it provides chan-

* The acronym "POLIS" stands for "Political Institutions Simulation." The POLIS Lab is a state-of-the-art facility for social gaming and simulation (see Robert C. Noel, "The POLIS Laboratory," AMERICAN BEHAVIORAL SCIENTIST, vol. XII, no. 6, (1969), pp. 30-35). The Lab was established under funding from the Regents of the University of California, the National Science Foundation (GY-6345), and the Chancellor's special funds at UC Santa Barbara. This support is gratefully acknowledged.

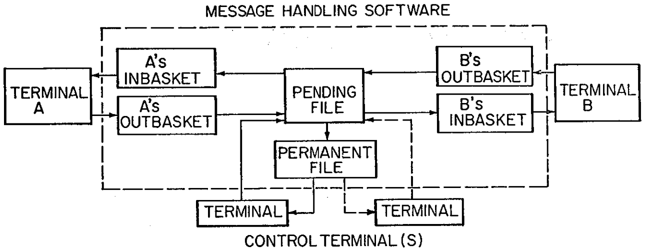


Diagram 1—*Message-handling: Manual intercept mode*

nels for written communications among player teams. Physically, the teams may all be at one site, or they may be dispersed at remote locations. In this sense, the system is essentially the same as a computer controlled teletype network. Second, the system provides control capabilities for the game director(s), including the capability to define specific communications networks, the capability to monitor communications content, and the capability to maintain complete game data files. Third, the system interfaces both game players and control staff interactively to man-model simulation programs.

In the following sections, the component programs of the system are briefly described. These descriptions are organized around two perspectives, that of the player and that of the game control staff.

Player oriented programs

The program modules described in this section perform a variety of functions for the players in a gaming exercise. They include a message handler, a text editor, and a simulation program interface module.

Message-handling

The message-handling programs in the system are designed according to an "inbasket/outbasket" principle. The system accepts a message from a user terminal when a user request is made for "OUTBASKET." The message is routed to an intervening file structure where it stays until its recipient makes an "INBASKET" request; at that time the message is printed on his terminal. These input and output events may occur at any time; they may be nearly simultaneous or hours or even days apart. How often a player interrogates his INBASKET may be left to his discretion or scheduled as part of a game's operating procedures. Straight-through, real-time communications will also be possible as a result of scheduled system revisions.

The message handler has capabilities for three modes of operation: the manual-intercept mode, the automatic mode, and a mixed mode. In addition, it embodies flexible addressing capabilities.

Message-handling: Manual-intercept mode

Diagram 1 depicts two player terminals interacting within the framework of the message handler. The routing of a message, say, from terminal A to terminal B is as follows. The message is input via A's OUTBASKET, which, in effect, stores it in a "pending file." It will remain in the "pending file" until it is acted upon by a control terminal. The system permits two or more control terminals to have this function; one or more might serve as an umpire terminal and another as an observer terminal. As will be discussed below, an operator at a control terminal can do three things to a message. First, he can reject it as is; that is, the message will be returned to A by way of A's INBASKET. Second, he can forward the message as is. Third, he can forward or reject the message after making editorial comments and changes on it.

Diagram 1 also depicts the fact that each message is entered into a permanent game file automatically as it is handled.

The idea of having control terminals as part of the communications channels stems primarily from the requirements of free-form games in which judgments by the game control staff (umpire) play an important role in the game. It also pertains to messages and documents sent from player teams to the game control staff, for example, position papers and moves.

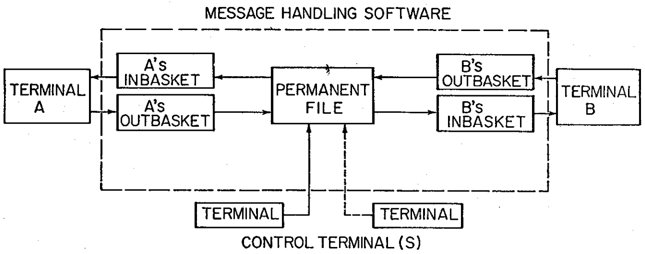


Diagram 2—*Message-handling: Automatic mode*

Message-handling: Automatic mode

When the message handler is set for automatic mode, all messages are routed to their destinations without being intercepted by control terminals (Diagram 2). As with the manual-intercept mode, each message is entered into a permanent game file automatically as it is routed through the system.

Message-handling: Mixed mode

The system provides the capability of having some sets of teams linked together in the automatic mode and other sets of teams linked in the manual-intercept mode, all within the same gaming exercise.

Message-handling: Addressing options

The message-handling software also provides the user with several addressing options. Of course, he may address a message to any other team in an exercise and to the control terminal(s). Additionally, he may input messages with multiple addresses; the system will route them accordingly. At the discretion of the primary control terminal, a particular exercise may include collective actors with group names. For example, the address "Security-Council" may include all members of the Council who are represented by teams in the game; each member of this group will receive the message accordingly. A special case of a group name is "ALL," in which case all active terminals will receive the message.

The system is able to accept player team names in natural language; it does not require numerical designations for the teams.

Message-handling: formatting

The system also embodies a few features which are designed to assist the participants in exercises in maintaining their own records. One such feature is the "paginator." At the present time it has been implemented only for messages being retrieved from one's inbasket. It prints each message on a separate page, which it measures to eleven inches in length and numbers in serial order. Anyone who has attempted to maintain order among the myriad scraps of paper which accumulate in a busy game will appreciate this feature.

Another aid to the participants and to one who is trying to reconstruct game events after the game is over is the "date/time clock." It prints a calendar date and zulu time on all incoming and outgoing messages. The clock may be set to print actual times or any past or future times desired by the game control staff.

User text editing

A second set of player oriented capabilities complements the message-handling programs. A text editor will be made available to player terminals in the future revision of the system. It will serve as an aid in the on-line composition of messages. It allows the user to delete and rewrite all or part of the line of text on which he is currently working and to delete and rewrite the entire message so long as it has not become input to the system through an "END" command.

With the scheduled revisions, the system will also accept paper tape input of messages composed off-line.

Player interaction with simulation models

A third set of player oriented capabilities allows the user to call a particular simulation program from a simulation program library. The user's request is "LIBR." The system responds with an interactive version of the particular man-model simulation called from the library. The player is given data upon which to base his decisions and the simulation program leads him conversationally through the decision sequence ending with a printed report of the results of his decisions. From each team's point of view, this capability is very much like having access to a time-shared system which computes only its portion of the simulation model. Game procedures determine the timing of these interactions. The model itself is initialized and updated by the game control staff.

Control-oriented programs

A more extensive set of control capabilities have been programmed into the POLIS game-administration system. They include programs for defining and modifying particular gaming communication structures, programs for monitoring and editing player interactions, and programs which assist in the reconstruction and analysis of games after they have been completed. These capabilities are described briefly in the following sub-sections.

Creating a game

Both logically and temporally, the primary control oriented capabilities of the system have to do with the creation of a game. These capabilities are embodied in the system's "CREATE" functions. These CREATE functions are the only control functions that cannot be delegated to any participating entity other than "POLCON," (POLIS Control), the primary control terminal.

The CREATE function enables POLCON to do the following.

(a) Set the date/time clock for the game.

(b) Specify up to forty-eight participating entities (player-teams, control teams, etc.) in a game by proper name and user code. These names may include group names, including specification of which teams belong in which groups.

(c) Restart from a previous game (the same game, in case of system failure), which eliminates the need to recreate the game.

(d) Assign functions to participating terminals, including delegating control functions. For each entity in the game, including both player terminals and control terminals, POLCON may designate:

    (1) whether it is to be able to send messages (outbasket function) and/or to receive messages (inbasket function);

    (2) whether its messages will be handled in the automatic or the manual-intercept mode;

    (3) whether it is to be able to "CALL" Control in real-time, bypassing the outbasket/inbasket message-handler;

    (4) whether it is to be able to call "LIBR" in order to interact with a particular man-model simulation program; and

    (5) whether it is to have such control functions as: "STATUS," "RETRIEVE," and "UMPIRE" (described below).

Updating a game

Another control oriented program complements the CREATE functions. It is an UPDATE program. It permits the dynamic modification of most CREATE functions, including the addition of new participating entities (by proper name), resetting the date/time clock (forward or backward), and the reassignment of message-handling functions.

"Control's" message management functions

When the system is in manual intercept mode, several message management capabilities are at the disposal of the game control staff.

(a) A "STATUS" request enables the game control staff to determine the contents of the game file for any time period and any set of participating teams. All messages are listed by time, parties, and subject matter. Their status is noted, that is, whether they are pending or whether and at what time they were forwarded or rejected.

(b) A "RETRIEVE" function is also available to the control staff. It extends monitoring capabilities to enable complete messages to be examined after they have been identified via a STATUS check.

(c) Control terminal(s) can also perform "UMPIRE" functions. These pertain to any message in the pending file. They include "EDIT" and "FORWARD/REJECT" options. That is, messages may be forwarded without alteration; they may be commented upon and edited (deletions, insertions); and messages and papers may be returned to their points of origin.

"Control" interface to simulation library programs

From a control point of view, simulation interface primarily involves calling up the appropriate simulation model from the program library, initializing the model's variables and parameters (analogous to, but different from the CREATE function), and establishing files for recording time-series data on the model in operation.

"Control's" data-making functions

The system also provides some textual analysis capabilities for the game control staff. Supplementary control terminal(s) may be established from which observer(s) may engage in real-time content analysis and commentary. This is done through a part of the umpire function. The request, "CODE," elicits a system response providing the observer with up to six empty coding categories which he may use to define content dimensions (for example, a threat) and to assign numerical rating for a particular message on that dimension. The request "COMMENT" enables the observer to append open-ended commentary to a message—perhaps an explanation of his coding judgments, or perhaps some information about the context surrounding the particular message. Neither observer coding nor observer commentary information is seen by the players; both kinds of information become part of the game file.

SYSTEM ARCHITECTURE

Hardware configuration

The POLIS system for the administration of gaming exercises was designed and programmed to execute in a hardware environment consisting of the following:

(a) PDP 11/20 CPU (with 60 Hz Clock Interrupt)

(b) 8K (16 Bit) Words Core Memory

(c) 2 Magnetic Tape Drives (Dual "Dectape")*

(d) Primary Teletype Console

(e) Alphanumeric Television Display

(f) 2 Acoustic Couplers with telephone answerers and associated control and interface logic.

Although the current system executes in the above configuration, extensive modifications are under way to take advantage of a recently acquired 64K word fixed head disc. This will result in a revised version of the system which will have features identical to the present version, but will eliminate some long interaction delays (up to 30 seconds) currently caused by frequent accesses to magnetic tape.

There is no limit to the number of acoustic couplers which can be simultaneously accommodated by the software. The number specified above reflects a core memory limitation only, that is, each additional 4K words of core above the basic 8K would allow eight more couplers to be accommodated. This is a result of the fact that 512 contiguous words of core memory are required for each coupler.

System-management software

Hardware drivers

Two peripheral-device drivers are employed by the system to manage asynchronous, interrupt-controlled I/O, a teletype driver and a magnetic tape driver. Both drivers operate in conjunction with a TASK SWITCHING PROGRAM (discussed below) to initiate an I/O transfer, to set the appropriate interrupt status bits, and to relinquish control to some other user. When the I/O transfer is complete, a hardware interrupt returns control to the particular driver which then processes errors if necessary, sets the "I/O Done" bit associated with this user and returns control to the TASK SWITCHING PROGRAM. Both drivers can manage I/O transfers simultaneously for any number of users.

Initialization and system restart

These programs operate together to initialize the following information for the system after either an initial start-up or a start-over from a system or power failure.

(a) An entity called "POLCON" (POLIS Control) is specified and allocated all of the available system functions.

(b) A Task Control Block (TCB) is created for the primary teletype console and each acoustic coupler.

(c) A Master Reset is executed to remove all interrupts pending, and control is passed to the USER REQUEST DISPATCHER.

This minimum system state is sufficient to allow POLIS Control (POLCON) to log-in and create a new game environment or reestablish the environment that existed before a system failure.

Task switching program

This program "time-shares" the use of the system among all of the I/O drivers. Control is relinquished to the TASK SWITCHING PROGRAM, which then checks sequentially the "Done" bit in each Task Control Block (TCB) and transfers control (back) to the first user it encounters whose "Done" bit is set. Thus, the following sequence of activity occurs (assuming, for example, that only two consoles are active).

(a) The program being executed by User "A" requests an I/O transfer.

(b) Control is transferred to the appropriate driver which clears the "Done" bit in the TCB associated with User "A's" console and the I/O transfer is initiated.

* Copyright Digital Equipment Corporation.

(c) Control is transferred to the TASK SWITCHING PROGRAM which then checks the "Done" bit in each TCB until it finds one that is set. If the "Done" bit in the TCB associated with User "B's" console is set, control is then returned to the program being executed by User "B."

(d) While User "B" is executing, the I/O transfer requested by User "A" is terminated under interrupt control and the "Done" bit in his associated TCB is set. When the program being executed by User "B" requests an I/O transfer, steps 1-3 again take place and User "A's" program will again gain control.

Since the system is inherently I/O-bound, this method permits each user to consider the system to be dedicated to him alone.

Clock interrupt handler

This program maintains an internal calendar/clock which is initialized and updated via "CREATE" and "UPDATE" functions.

User-interactive software

User request dispatcher

This program permits a user to log-in and execute those system functions which he requires.

The user initially gains access to the system by typing in "CTRL/P". This program then requests that the user enter his game name and two-character protection code. After validity-checking this information, the program then asks the user for the system function he wishes to execute and, if this user has been allocated this function by POLIS CONTROL, control is passed to the requested program.

All of the user-interactive programs discussed immediately below are executed as subroutines of the USER REQUEST DISPATCHER.

Message switching programs

The primary function of the POLIS system is to route messages from one user to another. Each message entered into the system is recorded on magnetic tape before being routed to its destination.

(a) *Send Message Program.* This program allows a user to enter a message into the system to be delivered at a later time to another user. The program first asks for the address of the message. This may be a single name, a group of names, a special name which represents a group of names, or all players in the game. Next, the user is asked to enter the text of the message.

The program then records on magnetic tape the addressing information, the text, and the time that the message was input.

A particular user may operate in one of two modes: manual or automatic. In the former, any message input by the user will be given a pending status and will not be made available to its recipients until POLIS CONTROL has taken a specific forwarding action. In the later mode, any message input by the user will be made immediately available to its recipients.

(b) *Receive Message Program.* This program allows a user to receive all non-pending messages addressed to him since the last time he executed this program. The program prints out the body of the text as well as the time it was received.

(c) *Console-to-Console Communication.* This function is not implemented in the present version, but it will be made available in a revised version. The purpose of the program is to link POLIS Control's (POLCON) console with that of some other user in order either to carry on a telephone-like conversation or to monitor the user's inputs to the system.

Control programs

(a) *Create Game.* This program allows POLIS Control initially to specify the entire environment for a game. The environment includes the following information.

(1) the date and time

(2) the simulation names of all users in the game

(3) the two-character protection code associated with each name

(4) any special group names (i.e., names which represent groups of users)

(5) the system functions which each of the users will be allowed to execute during the course of the game

This information is stored in the system tables discussed below. The program also makes a copy of this information on magnetic tape to permit a recovery in case of a system failure or to run an identical game at some later time. This function cannot be allocated to any user except POLIS CONTROL.

(b) *Update Game.* This program allows a (control) user to make dynamic changes in the game environment (such as the date-time). The game environment may be changed in any way, with the exception that names may not be deleted.

(c) *Message Tape Status.* This program allows a user to get a report of the messages stored on the magnetic tape. The report may be requested for any time period from the beginning of the game to the present, for any subset of senders, and/or for any subset of recipients. The report consists of the following information about each message.

(1) the sender
(2) the recipients
(3) the time sent
(4) the status of the message (i.e., pending, forwarded, or rejected)
(5) the sequential number assigned to the message by the SEND program
(6) the capsule summary of message content

(d) *Retrieve Messages Program.* The execution of this program is identical to that of the message Tape Status program except that, in addition to the information reported, the entire text of the messages is printed. This function is used in conjunction with the program discussed next to allow a controller to effectively monitor the course of a game.

(e) *Umpiring Program.* This program provides the means for a controller (such as POLIS CONTROL) to monitor the course of a game and to make significant modifications to the messages that are routed between the participants in the game. Specifically, the controller is allowed:

(1) to forward a message to its recipients or, if the message is deemed to be inappropriate, reject it back to its originator;

(2) to delete and/or insert text in the body of a message;

(3) to encode a group of categories pertaining to the content of a message; and

(4) to append an opaque comment about the message to be used in a post-game analysis of the game.

Whenever a message is modified in any way, a copy of the original is retained in order to maintain the continuity of the game for subsequent analysis.

POLIS network explanation

This program allows a user to interrogate the system to get a brief description of any or all aspects of the system.

Non-system software

The POLIS system is designed to allow execution of non-communication system programs in either of two modes; stand-alone or interactive.

Stand-alone (Library) programs

In this mode, man-model simulation programs which require the facilities of the entire computer are stored on a library tape and, when requested, preempt the execution of the system. A core image of the system is saved and the requested program is swapped into core. After completion of the program the system is restored and any game which was in progress can continue as though no break had occurred.

Interactive programs

In this mode, simulation models may interact with an on-going game in much the same way as a live participant. That is, inputs to and outputs from the model will be stored on the message tape as though they were actual messages. This function will be implemented in the 1972 revisions of the system.

Data-base structures

Task control block (TCB)

The TCB is a table of 512 core words which contains all of the information associated with a particular user console. In the current configuration there are three TCB's, one for the primary teletype console and one for each of the acoustic couplers.

The TCB contains the following information.

(a) a pointer to the next TCB
(b) the contents of the stack pointer (relative to this user's stack) at the time this TCB last lost control due to an I/O request
(c) this user's stack (64 words)
(d) the program restart address for this user
(e) the device number associated with this TCB
(f) the status of the I/O device associated with this TCB
(g) an input buffer (72 bytes)
(h) items for the address of a message to be output and the length of the message
(i) Dectape status
(j) Dectape block number to be input/output
(k) Dectape buffer (512 bytes)
(l) the entry number in the name table of the user currently using this TCB
(m) work space for the various programs which this user may execute (139 words)

System tables

The system tables contain all of the information about the current game which was initialized by the Create Game program.

(a) *Permanent Bit-Map,* which maintains an account of the currently available blocks on the message tape. There are 576 blocks of 512 bytes on a Dectape. A zero bit indicates that the associated block is available. A one bit indicates that the associated block has been used.
(b) *Date and Time calendar/clock,* consisting of twelve numerical ASCII characters which represent the date/time (formatted as: 11/1/71-09:30:30).
(c) *Name Table,* consisting of a packed table of ASCII characters which comprise the names of all of the users in the game. The names may be composed of any ASCII characters and may be of any length.
(d) *Protection Codes* (two characters), associated with each name in the Name Table.
(e) *Privilege Table,* consisting of a table of 16-bit bitmaps indicating the system functions which have been allocated to the associated name in the Name Table.
(f) *Subordinate Name Table,* which links the entry numbers of group names with the entry numbers of the participant names which they represent.

Magnetic tape files

Files on the magnetic message tape are composed of linked blocks. The first word in each block is a pointer to the next block in sequence. Files may be of any length, and a block-pointer of Zero indicates the last block in the file. Consecutive blocks in a file are physically at least four blocks apart. Each file contains the image of one message.

The Tape File Directory consists of ten blocks on the tape and is composed of 226 entries of eleven words each. Each entry contains the following information about the message contained in the file.

(a) the number of the first tape block in the associated file
(b) the status of the message (e.g., PENDING)
(c) the Name Table entry number of the sender
(d) the date/time that the message was sent (coded as twelve 4-bit segments)
(e) a bit-map indicating which of the original recipients have not yet received the message

SUMMARY

In this paper we have tried to identify the main characteristics that should be incorporated into a comprehensive, computer-based system for the administration of social gaming exercises. We have described, first functionally and then technically, the game administration system developed at the POLIS Laboratory. Our system design combines information management, simulation, and data structuring and analysis capabilities into an integrated, user-oriented package. A limited, first version of the system is currently operational on the Lab's PDP-11 system. Realization of the full design is our development goal for 1972.

ACKNOWLEDGMENTS

The authors would like to thank Messrs. Robert Dolan and Carl Magagnosc for their invaluable contributions to the programming effort described in this paper. In addition, we gratefully acknowledge the assistance of

Mr. Michael Spafford, the POLIS Lab's engineer, who kept the computer hardware operational when needed.

The senior author wishes to acknowledge his indebtedness to Gerald H. Shure and his associates at the Center for Computer-Based Behavioral Studies at UCLA. His association with that group, which included Drs. Robert Meeker, Harvey DeWeerd and Richard A. Brody, led to the development of the work described in this paper.

REFERENCES

1 R R BEATTIE L BLOOMFIELD
Cascon: Computer-aided system for handling information on local conflicts
ACDA-WEC-141 Center for International Studies MIT #C/70-8 December 1969

2 Shure's gaming communications system was operational in 1968, on the System Development Corporation's Q-32 computer, which has subsequently been phased out of service

The development of process control software

by JAMES D. SCHOEFFLER

Case Western Reserve University
Cleveland, Ohio

INTRODUCTION

The use of on-line, real-time computers for control of industrial processes has been increasing rapidly during the past ten years. That the cost of the software necessary to implement such systems exceeds even the hardware costs became clear in the initial installations. As a consequence, much effort was devoted to the development of efficient, economical software and software approaches for use in industrial process control applications. During the past five years, there has emerged from these efforts the realization that process control software is different from software for large scale batch processing, time-sharing, message switching, or any other computer applications. It contains its own requirements and problems and leads to a distinct set of solutions. Moreover, this difference is due to more than the size of the computers involved in industrial process control. The objective of this paper is not to catalog the significant features of various software systems in existence today for this has been done very well in a number of recent survey papers. Rather, the objective is to describe the basic structure of current industrial process control software, emphasizing the unique structure of that software, how it evolved, and its current points of controversy and problems.

THE PROCESS CONTROL APPLICATION

The source of the uniqueness of process control software is due to the nature of the application, environment, vendors, and users. There is a large variety of processes in the paper, rubber, chemical, petroleum primary metals industries. Control of such processes generally involves a set of plant instrumentation including sensors, transmitters, and special instruments which provide measurements of physical and quality variables important in the process application. Such measurements may be continuous in time (temperatures, pressures, flows, thickness, speed) or discrete in time (concentrations of a stream as measured by an on-line chromatograph from periodic samples from the stream, average cross machine basis weight as measured by a scanning beta gauge, quality variables which are measured off-line in a laboratory). Process control involves the use of these measurements in order to help the human operator more economically run the process.

This objective may take one of several specific forms. For example, regulation or direct control has as its objectives the maintenance of critical process variables at prescribed values called operating or setpoints. The need for regulation of pressures, temperatures, speeds and the like, are obvious. Quality variables too may be regulated as for example maximum moisture in a sheet of paper or thickness of sheet steel. Regulation involves the basic feedback process: the comparison of measured and desired values (set points) to produce an error and the use of that error signal via a control algorithm to change some manipulated variable by a physical actuator. For example, the regulation of tank liquid level might be carried out by manipulating the position of a valve in an input stream. Such control is relatively fast, involving sampling of hundreds of variables several times per second, and outputting to actuators at the same rate. The periodic nature of this task is one of the dominant features of industrial process control in that the time sharing of the computer among many such tasks results in critical response requirements on the hardware and software.

Control is a tool rather than the end in itself and necessarily is modified from time to time as the process needs or physical structure change. The application, however, does not permit the taking of the computer off-line for program preparation but rather demands that the software be capable of modification while the process is running on-line with attendant guarantees and assurances of safe reliable bug-free operation. Hence, an essential ingredient in process control applications is operator integration which takes place through special purpose consoles oriented toward the particular application. From these consoles, an operator may examine any variable in the system, produce summary logs of the operation of the process, demand changes in parameters being used for control of the process (setpoints for example), and may even call for changes in the structure of the control system: adding variables to be scanned, adding control calculations for regulatory purposes, or deleting control loops. This implies that all of the parameters and data associated with the process control regulat:on program must be in a form which can be mnemonically referenced and modified by an operator. Thus, the data base associated with regulatory control cannot be imbedded in the programs. This leads to the special structure of this software.

Supervisory control differs from regulatory control in that the objectives are not the maintenance of process variables at particular values but rather the determination of those operating or set points. For example, operating guides are operating conditions which have been deduced during previous successful operations of the process. When similar conditions hold, the computer directs the operator to use these operating points or else justify any deviations. This takes away the option of operating the process in a safe but uneconomical manner.

Optimization of operating conditions may be the goal of the supervisory control system. Optimizing control is the determination of set points from an analytical performance criterion plus a mathematical model of the process. Successful optimal control has been carried out in the petroleum, steel, and electrical power industries among others. Supervisory control is characterized by much more complex programs than regulatory control but carries with it the same demands on reliability, centralized data base, and operator communication. That is, mathematical models of the process imply knowledge of the process state which in turn implies knowledge of all the variables, scale factors, offsets, calibration factors and other parameters which make up the process data base. The operation or status of every instrument, actuator, and control in the process is needed to determine the current state of the process. All of this information must be made available to the operator on demand and so it too must be organized in a central, accessible data base.

Many special purpose programs are necessary to support the basic application including calibration, hardware checkout, special logging and data reduction programs, startup and restart routines, and general programs to maintain and update the data base, including tables of mnemonics. In addition, the functions which are not automatically scheduled according to time or event information must be available on demand of the operator, implying an on-line, interactive command interpretive program. All of these many programs must co-exist, interact through the data base and executive control programs and hence lead to a far from trivial implementation problem. The result is the current process control software organization described in the following sections.

PROCESS CONTROL SOFTWARE

For a very brief interval, it was thought that process control software would, after initial debugging, run without change for a long period of time. If this were true, many of the problems of the initial systems would not have existed and the current structure of process control software might be radically different. The process changes often, necessitating similar changes to the software. The chief characteristic of modern process control software is its ease of change, permitting on-line modification of control algorithms and even addition and checkout of programs while the system is running. The typical computer used for industrial process control during the last five years is what today would be considered a medium-sized machine, significantly larger than today's minicomputer in the sense of attached peripherals. As a consequence, it was necessary to provide an executive control program to oversee the multiple interacting programs operating in a real-time, concurrent manner. This need, plus a desire for ease of modification and the demand for a flexible, general process data base leads to the structure shown in Figure 1.

That figure shows a conventional real-time executive control program (a monitor) along with a process data base and application programs. The distinctive part which makes it uniquely process control software is the set of application packages which are designed to carry out specific tasks in a very efficient manner while at the same time permitting easy communication among themselves, the executive, and the application programs.

Process control executive

Successful process control involves the implementation of many concurrent tasks. Process control programs doing data acquisition, control calculations, analog and digital output, operator communication and the like call for some multiprogramming capability. Executive control programs recognized this and incorporated the multiprogramming in one form or another. Early executives organized core into two partitions, one containing

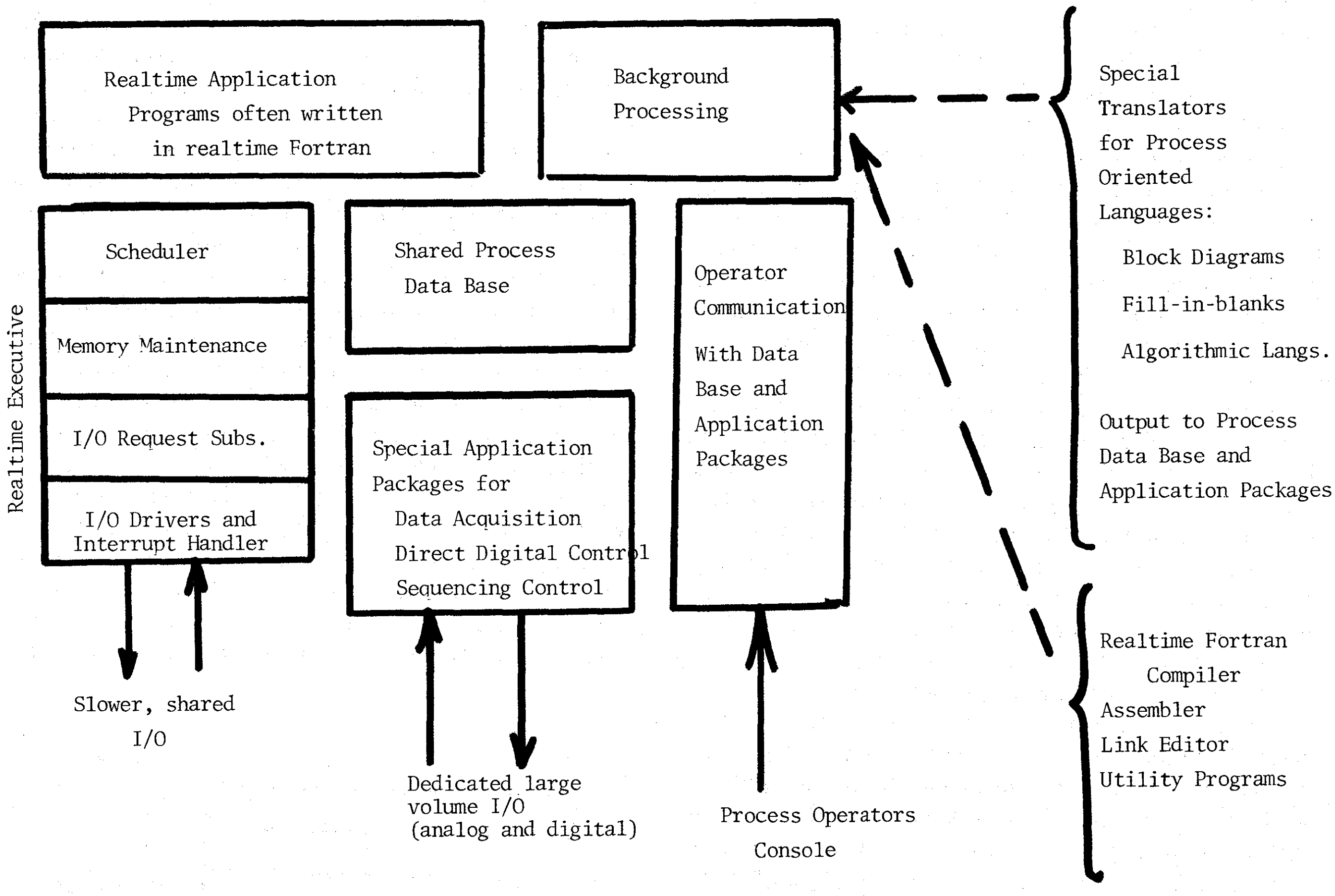


Figure 1—Structure of process control software

permanent resident programs (those requring fast response or maintaining control system integrity in case of failure of the bulk storage system) and the second being used to swap program modules called coreloads in and out of core. A coreload consists of one or more mainline application programs plus subroutines used by these programs and not contained in the permanent resident software. Coreloads may have different priorities in which case the executive often interrupts execution, swaps the coreload out (saving it on bulk storage), swaps in and executes the higher priority coreload and later restores execution of the interrupted core load.

Multiprogramming based on coreload swaps proved too slow for many applications. This led, of course, to a need for large and larger core storage so that critical programs could be permanently core resident. This motivated a change toward a more dynamic allocation of memory. The most common system is a multipartition system in which all programs assigned to a partition run with the same priority. This implies that a program, once loaded into a given partition and execution begun remains in that partition until its task is completed. Then the next scheduled task for that partition is swapped in to replace the completed program.

The partition system recognized a perhaps nonobvious fact of life of process control systems: the critical resource is not main memory but rather the bulk storage channel (usually the disk). This is especially true of systems based on moving head disks of course. Running tasks to completion minimizes the number of bulk storage swaps while maintaining the priority levels of tasks.

The multipartition system requires the allocation of a program or task to a partition and the assurance that all such programs fit in a given partition and have the same priority. This requires a careful analysis of the application but results in a smooth, controllable realtime implementation.

Most machines today are of the 16 or 24 bit word length variety. The 16 bit machines in particular take advantage of the partition system because there is no need for dynamic relocation of programs. The 24 bit machines can relatively address ±8K of core, a value sufficient to permit dynamic relocation of most process control programs (but not of the data base). In some executives, core is divided into partitions with some of the partitions being used as indicated above but with one dynamically allocated by the executive. The latter facilitates background Fortran-level processing.

Process control executives contain extensive error recovery capability because of the difficulty of restarting the system in case of a crash with critical data in core. Error recovery includes backup of I/O devices where feasible, specification of alternative devices, multiple tries at attempting reading of bulk storage, automatic power fail program backup, and automatic maintenance of copies of critical files like the process data base.

Such executives permit multitasking in one form or another but not usually in the completely general case. That is, tasks in a partition are permitted to schedule (turn on, turn off, delay, or queue) other tasks in that and other partitions. However, the allocation of the task to core is predetermined and its priority is not dynamic, but rather pre-assigned. This has proved necessary not only because of the difficulty of dynamic relocation but also because of the need to guarantee a maximum response time for critical real-time programs.

I/O in these executives is handled in the straightforward manner, using I/O drivers and I/O request subroutines. Drivers are responsible for the maintenance of the interrupt driven I/O with the actual device, outputting or inputting one character or word at a time usually with one interrupt per data item transferred. Little I/O is done through direct memory channels except in the case of bulk storage devices. The I/O request subroutine is used to control the competition among many tasks at different priority levels from interfering with one another. Such routines queue requests, notify routines (reschedule them) when a requested I/O operation is complete, test device status, etc.

As long as the volume of I/O is low or the response time of tasks not critical, such an organization is quite sufficient and essentially like any other application. That is, none of this software is unique to process control.

Two situations arise, however, which call for unique process control software. They are: the need for high volume, very flexibly organized process I/O; and the need for mnemonically oriented operator communication. These require special packages for their implementation and are discussed below.

Special packages for data acquisition and control

The need for generality and flexibility in the typical industrial data acquisition system is paramount. Not only is it desirable to be able to change the parameters of the data acquisition loop or the status of a sensor from active to inactive, but also to be able to add and delete from the list of points in a flexible fashion. Thus, if a sensor is added to the system, it is desirable that the operator be able to add the point to the list of converted values, including all the appropriate parameters necessary for the conversion and all the linkages to control programs associated with the variable out of limits, bad data, and so on. This, of course, can be done by reprogramming the routine, but is undesirable and not necessary. The basic operations involved in converting a point are similar from point to point. The scan routine uses the address of the point, values for the amplifier gain, limits, etc., and in effect, applies similar sequences of algorithms to each of the points to be converted. A data acquisition package takes advantage of this as shown in Figure 2 which shows an interpretive scan program and a set of data structures called loop records, with one loop record associated with each point to be converted. All the data and parameters associated with a particular point is stored in the record, including the

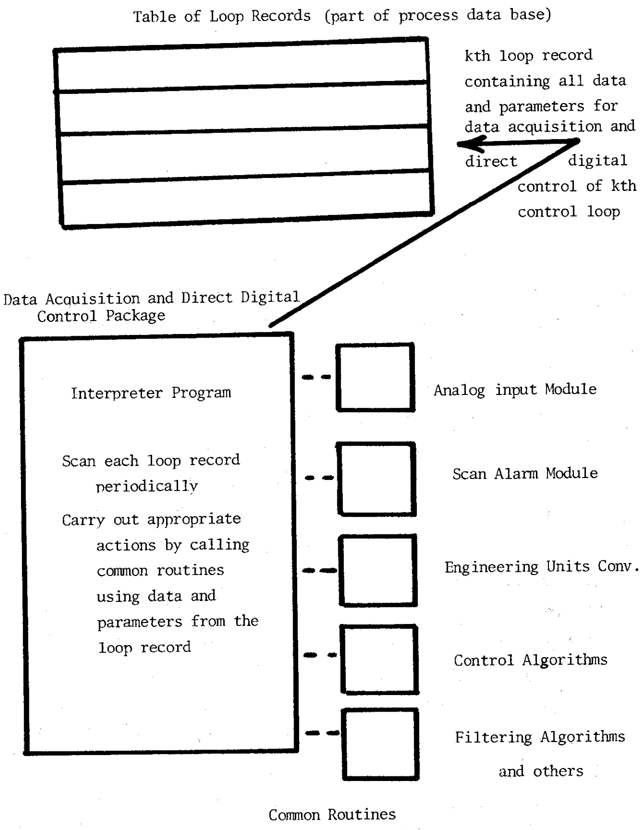


Table of Loop Records (part of process data base)
kth loop record containing all data and parameters for data acquisition and direct digital control of kth control loop
Data Acquisition and Direct Digital Control Package
Interpreter Program
Scan each loop record periodically
Carry out appropriate actions by calling common routines using data and parameters from the loop record
Analog input Module
Scan Alarm Module
Engineering Units Conv.
Control Algorithms
Filtering Algorithms and others
Common Routines

Figure 2—Example of data acquisition and direct digital control package

period or time between conversions, status of the point, external names, etc. The interpretive program scans the records in some sequence until it finds one which is scheduled for conversion. It then examines the loop record to determine the appropriate data and algorithms to be carried out on that data point. Pointers to the individual algorithms to be carried out on the point, followed by the parameters to be used by that algorithm, are stored in the loop record. If any error is detected in the conversion, an error recovery routine is entered and executed. This might include an attempt to reconvert the point or to estimate the current value in terms of the previous value (which is stored in the loop record), or preparation or notification of either a program or the operator. The interpretive program then continues through the loop records, is offset and scaled. After conversion to engineering units, the data is limit checked for validity, alarm checked, and perhaps digitally filtered by a moving average filter involving the current value and the previous several values. The set of loop records is a very well organized process data base. The loop record concept is a very flexible one in that the actual algorithms carried out on any particular data point may vary considerably by simply changing the parameters and pointers in the loop record.

The only portion of the program particular to a specific set of data points to be read is the information stored in the loop records. It follows that the interpretive scan program can be written and debugged without actual knowledge of a particular application except that all algorithms which might be needed in the application must, of course, be provided. Individual points can be added or deleted by creating or deleting one of the loop records. Because of the fixed format of the system, it is possible to organize the various loops so that not all points with the same conversion period need be carried out at exactly the same instant of time. That is, points with the same conversion frequency can be subdivided into smaller groups which are converted at the correct frequency but offset or phased with respect to one another in time.

The efficiency of such a system must be considered. The interpreter scans through items in a loop record and in effect calls small modules or sub-routines to carry out the desired operations. There is a certain amount of overhead associated with scanning through the loop record, determining which routine is to be carried out, transferring to this module or routine, as well as extracting from the loop record the parameters and data needed by that algorithm. If the module or algorithm is too small, it follows that this overhead involved in scanning, transferring and extracting parameters, may be comparable to or even large compared to the time to carry out the algorithm. The result would be excessive overhead, for inline code to carry out the same task would be more efficient. The objective is, of course, to use modules whose execution time is large compared to the overhead or scan time so that no real overhead penalty is incurred. The overhead associated with interpreting the loop records is offset by the flexibility of the system. Most important is the ease of operator communication with the data base.

If this can be done, of course, the generality and flexibility of the interpretive loop record organization can be achieved without discernible overhead penalties. Whether or not this can be done is dependent upon the organization of the loop record, the instruction set of the computer, and the addressing modes available in the computer.

Control loops lend themselves to a similar loop record organization. Additional functions (filtering, control algorithms, error checking, outputting, etc.) are listed in loop records often along with the data acquisition information and interpreted by the same or a similar routine. The result is a separate data base and data acquisition and control package which may be interrupt driven independent of the executive or possibly assigned a high priority level and controlled by the executive.

The special purpose packag? relieves the application programs from considering the data acquisition and direct control task in great detail including error checking, organization of operator communication, and startup. Other applications are similarly best handled by special purpose packages. These include sequencing control—monitoring the state of the process and causing events to occur (turn on burners, valves, etc.), in the correct sequence and when prescribed conditions are met. Examples include batch processing of chemical processes, control of annealing lines, and the like.

Operator communication becomes a straightforward task with a data base arranged as described above. Interface routines to the data base provide application programs the ability to read or write any particular item in any record (including a full complement of error checking). Copies for backup and restart are available because of the centralized location of data as opposed to its distribution throughout application software.

Operator interaction through a console limits the amount of error checking necessary and makes a very acceptable system for use in an on-line situation with operators who are not trained computer operators. Function keys are the most common communication scheme with labeling in industry-sensible terms. Functions limit the complexity of the operator console support routines but nonetheless are nontrivial in size.

In larger systems, it is convenient to separate the data base into two parts, that which is essential to incore reliable operation in case of disk failure and the remainder which is then swapped in and out of core (perhaps on a single record basis) as needed.

In addition to special packages which are unique to process control, other application programs are provided which may interface to the executive and the special purpose packages. Optimization is an example, permitting optimal control of a process unit provided a mathematical model and sufficient measurements are available. These differ from other application programs only in that they are "canned" rather than written by the user in a higher level language. Typically, they interact considerably with the data base and its support routines.

SOFTWARE PREPARATION FOR PROCESS CONTROL

Current medium-sized and larger systems universally provide the ability to write application programs in higher level languages, most notably, extended Fortran. This is both to limit the level of expertise of programmers but also to permit control engineers to produce application software. With intelligent use of data acquisition and control data base packages provided by the vendor and the general process control real-time executive control program (including all its entries for scheduling, error checking, I/O control, etc.), higher level language programming becomes quite feasible, especially in applications which involve significant computation. Other applications are developed more efficiently via packages like the data acquisition or direct digital control packages described above. These are usually programmed in assembly languages and linked tightly with the executive. In some cases, they are assembled as part of the executive.

The net result in Figure 1 is an executive which does not control all I/O but only that which is shared among application programs. Special purpose packages maintain a centralized data base and perform specialized control functions. Application programs perform tasks on a somewhat slower basis, but interface extensively to these packages through normal system subroutines.

Because of the need for flexibility and change, on-line program preparation is needed in most process control systems. This takes the form of a background command processor, compilers and assemblers and library maintenance routines as well as link edit routines. Program preparation including debugging can be done in a controlled manner in one of the partitions, without upsetting the remainder of the software system.

In recent systems, macro assemblers have become available, even including libraries of system macros which permit a user to custom tailor the operating system software to his particular application (at least in the minicomputer process control systems). At present, few cross assemblers are available and even fewer cross compilers for software preparation on time-sharing or batch data processing systems.

Much more significant is the preparation of data inputs for the dedicated application packages. The data base contains all the information needed for the carrying out of these dedicated tasks. The operator console, with its function keys, provides on-line data entry, program change, control application modification, and the carrying out of operator commands. Off-line program preparation is through translators of one form or another. The natural language for the application might be a block diagram indicating the interconnection of the measurements, setpoints, filtering, control and feedforward algorithms. Specified this way, they must be translated to data to be stored in the internal data base for interpretation.

Many such communication languages have been developed in order to provide a means for process engineers who are not trained programmers to set up and maintain an application involving hundreds and even thousands of variables. Such systems specify the application in an English language fashion which is then translated to internal data storage. Others are of the "fill in the blanks" form which is a questionnaire which is completely self documenting (and 100% of the documentation) and which, when punched line for line on cards, provides input to a translator which generates interval entries in the tables of the various packages.

With these programs, an application can be put onstream very rapidly since the essential programming is done and debugged independently of the particular application. Moreover, the change from one control structure to another (one startup procedure to another, etc.) is very straightforward, amounting only to change of data but no essential reprogramming. There is little penalty incurred in changing the control software, even less than in the case of Fortran programming of application programs. As mentioned in the next section, this package concept is less economical today because of the introduction of minicomputers.

The other trend today is toward the use of "realtime languages" for process control, even involving an attempt at a worldwide standard. There is a difference of opinion as to the economics of standard process control languages. As noted in Figure 1, the bulk of process control software lies in the executive and the application packages. Application programs tend to be written to use these programs and hence are less expensive (the complexity of the real-time application is imbedded in the application packages and the executive). One school of thought advocates the need for a higher level system programming language for the economic and efficient production of the executive and the application package or at least the tailoring of these standard packages to a given configuration and application. The other school of thought argues that these should not be specialized or customized but be standard so that they can be supplied by any vendor (and at reasonable cost since they can be written off over a large number of installations). This school of thought then argues that the remaining cost is in the application programs and that a language ought to be adopted for this purpose (extended Fortran, or some modification of PL/1 or Algol are among the suggestions).

It appears that no economic study has actually been carried out to determine the savings which might result from either of these approaches. The latter case already provides extended Fortran so that only incremental gains can be expected. The former case is difficult to assess because it depends upon the need for changing of standardized software packages or adaptation of these to various configurations of computers. If it were not for the minicomputer, data could eventually be gathered. However, as indicated in the next section, the minicomputer is changing the economics of process control software significantly.

CURRENT PROBLEMS IN PROCESS CONTROL SOFTWARE

The development of process control software over the past five years or so was directed toward the organization of Figure 1. Recently, packages for data acquisition and direct digital control, sequencing control, batch processing, process optimization, model identification, graphic operator communication, and others were developed. If the use of medium-sized machines continued, this software would have resulted in significant decreases in costs of process control software implementation. Several problems, however, indicate that further development is necessary.

First, the use of a medium-sized machine is difficult to justify economically in many applications. The capability of such machines is such that many tasks can be carried out by a single machine. This, in turn, places severe demands on the executive and application software in order that the installation can be carried out smoothly. Moreover, the process itself must be large enough to justify such a machine and such extensive software costs.

The advent of the minicomputer with its low cost has changed the picture considerably. The lower cost of small configurations implies that several tasks need not be implemented in a single machine in order to justify the use of a computer for control. This, in turn, implies that perhaps the extensive software developments of Figure 1 are not necessary for any smaller applications. The result has been the use of minicomputers for many small applications which in previous years had used larger machines. This, in turn, implies that the number of systems to write off software package development has decreased considerably. Hence, larger systems with very sophisticated software packages have recently been more difficult to economically justify.

The minicomputer poses a set of problems which are considerably different. First, the varied configurations and applications do not always require complex executive or monitor systems. Consequently, there is less justification for the development of extensive executives on the part of vendors. The result is sometimes more software development for the minicomputer system than for a larger installation, robbing the system of its apparent economic advantage over the larger system.

A second source of difficulty is the multicomputer configuration. The use of minicomputers for dedicated applications still requires the coordination of the systems to achieve overall process control including scheduling and higher levels of control. Hooking together a number of small machines each implementing a portion of the application is far from trivial for there is still a need for a coordinated data base and operator communication even if the application is distributed over a number of machines.

A related problem affecting the development of software for smaller machines is the rather commonplace prediction that the bulk of sales for systems will lie in discrete parts manufacturing rather than process control. The software necessary for this application has not been widely developed and agreed upon but will certainly affect the organization and structure of future minicomputer process control software.

The current question in process control software is how to economically produce the special purpose real-time programs necessary for a low cost but efficient implementation in a stand-alone minicomputer. In addition, how can the advantages of packages for specific applications be gained since it is difficult to develop them such that they are compatible with the varied configurations and computer systems arising today? Perhaps the answer is a higher level systems programming language with program preparation on a host machine. Perhaps the answer lies in generalized software preparation schemes such as are used to generate execution programs for large data processing machines. Perhaps the answer lies in the use of sophisticated template macro assembly languages. Whatever the answer, the well developed process control software concept is today facing a period of rapid change similar to that of five years ago.

REFERENCES

1 H E PIKE
Process control software
Proc IEEE Vol 58 No 1 Jan 1970 pp 87-97

2 J C MECKLENBURGH P A MAY
An engineering comparison of process control languages
Dept of Electrical Engineering Univ of Nottingham

3 *Special issue on computer languages for process control*
IEEE Transactions on Industrial Electrons and Control Instrumentation IECI-15 Dec 1969

4 P R WETHERALL
Corall 66—A language for the control of real time processes
ibid 2 p T173 1969

5 D G BATES
PROSPRO 1800
IEEE Trans Indust Electronics and Control Instrumentation IECI-15 No 2 p 70 1968

6 S J BAILEY
Pushbutton programming for on line control
Control Engineering 15 No 8 p 76 1968

7 G W MARKHAM
Fill-in-the-form programming
Control Engineering 15 No 5 p 87 1968

8 T G GASPAR V V DOBROHOTOFF D R BURGESS
New process language uses English terms
Control Engineering 15 No 10 p 118 1968

9 R E HOHMEYER
CDC 1700 fortran for process control
IEEE Trans Indust Electronics & Cont Instmn IECI-15 No 2 p 67 1968

10 T L WILLMOTT
A survey of software for direct digital control
ISA Paper 68-834 October 1968

Future trends in software development for real-time industrial automation

by H. E. PIKE

General Electric Manufacturing & Process Automation Advanced Development Operation
Lynn, Massachusetts

INTRODUCTION

"The __________ is a desk-size, stored program computer. It has a medium-scale capacity and uses a single address system. The __________ has all the advantages of high component reliability, automatic operation, and ease of programming." [1]

These words are from the introduction of the operations manual for one of the first minicomputers—the "LGP 30 Royal Precision Electronic Digital Computer." The manual was written in 1959. In twelve years we have just barely begun to understand the impact of this size computer upon the industrial automation.

In this paper we will examine future trends for programming for process control and real-time applications of industrial automation. Although blind forecasting is exciting, we will take the more conservative approach of examining past history and our current environment for factors which will lead to future developments. A short definition of the scope of applications considered will be followed by examination of current technological and economic trends. A brief recap of the companion survey paper by Schoeffler[2] will serve to establish the current state-of-the-art and a survey of current active development and/or standardization activities will be followed by predictions for the future. These predictions will be made from the fundamental viewpoint that future developments are determined by three factors: the current state-of-the-art, current problem areas, and new technological tools available for reduction to practical use. An underlying assumption is that there is or will be sufficient economic motivation to undertake the developments required, with timing being the only uncertainty.

RANGE OF APPLICATIONS

It is necessary to be more specific about the range of applications to be discussed. Future trends in real-time industrial automation will be discussed, rather than the more encompassing possibility of all real-time applications. Not included in our considerations are the applications typically called command and control, teleprocessing and interactive management information systems (certain portions of our discussions will, of course, apply to these areas).

Typical real-time industrial automation applications span a range from direct numerical control to load flow control and economic dispatch in the electric utility industry. Also included are direct digital control of chemical processes, supervisory control of chemical processes, sequencing control on assembly lines, control of batch chemical processes and computerized numerical control (see References 3-13).

In direct numerical control a small, fast, dedicated computer is used to control a machine tool directly. Typical required response times are on the order of milliseconds. Load flow control calculations on the other hand involve large data-handling requirements because of large matrix inversions, and involve response times of minutes.[14]

In direct digital control the computer directly manipulates the positions of control elements such as valves without an intermediate mechanical control device. Direct digital control requires response times on the order of seconds to disturbances in the controlled variable.[15,16,17]

In supervisory control the computer is used to provide setpoints which are then actually controlled by hard-wired analog equipment. In this case, the real-time response capability of the control system is on the order of minutes.

We could go on, but the point is clear: there exists a tremendous range of application requirements in this area which we have labeled industrial automation.

PRESENT STATE-OF-THE-ART

Schoeffler[2] has assessed the current available tools for software development. To summarize, in practical

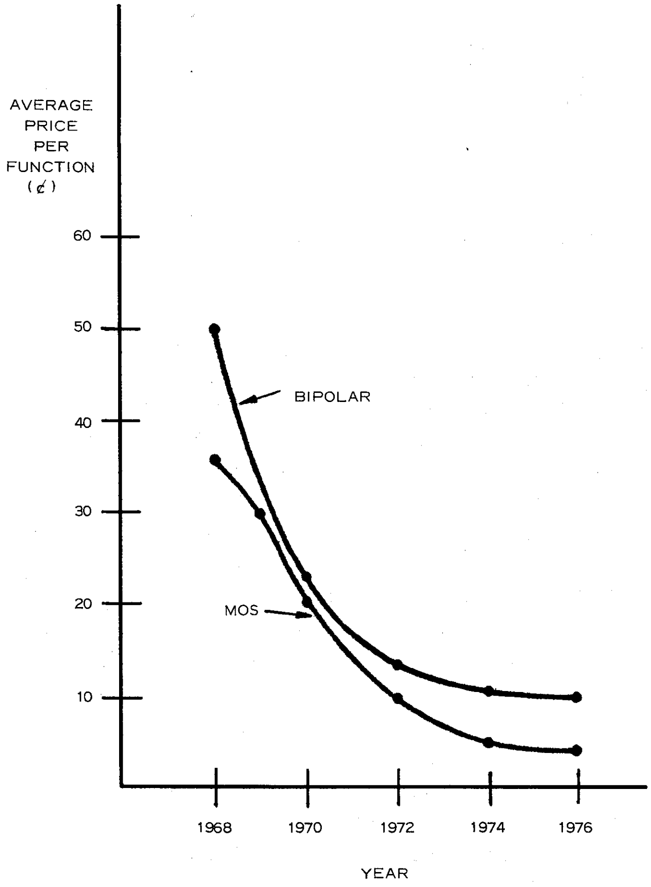


Figure 1—Digital integrated circuit cost trends

use today we have as wide a range of software approaches as there are applications,[18,19] about which the following statements seem to hold.

- Most of the real-time programming is still done in assembly language.
- Various dialects of Fortran have been tried, with mixed results. They have been most successful where a high content of engineering calculations is included (References 20 through 27).
- Initial experiences with problem-oriented programming systems appear very promising, particularly when two conditions are met:
  —functional similarity between installations, allowing a reasonable match between programming system and control problem (References 28 through 38).
  —sufficient economic base to merit development of general purpose packages.
- Various attempts at general purpose real-time languages have shown the concept to be feasible, but have remained as academic exercises, in part because of the inherent restrictions due to programming architectures of current computers (References 39-47).

CURRENT TRENDS

Two current trends will combine to motivate future development of improved programming techniques: microelectronics, and recognition of the cost and complexity of the software production process.

Microelectronics

The dramatic downward trend in price/function in microelectronics is widely recognized. Figure 1 is typical of the current extrapolations of this trend. An easily customized MOS/LSI desk calculator chip has been recently announced at a reported volume price of less than $20.48. Our colleagues in the hardware design area confirm that this device is comparable in logic power (in terms of gate equivalents) to the LGP/30 of ten years ago discussed above.

If this isn't enough, a major manufacturer has announced a CPU on a chip, of perhaps five times the complexity of the LGP/30, and our R&D associates advise us that such devices with electrically alterable ROM's are feasible, for the ultimate in flexibility.

The message is clear—Software functions will disappear into hardware (or firmware) and modest investments in increased hardware capability (if made correctly) will reap outstanding software cost savings.

The software development process

Another important trend is increasing recognition of the difference between "programming" and "software development." Examining the following breakdown of the software production process makes this distinction clear:

1. Problem analysis
2. System structure design
3. Logic design
4. Program coding
5. Language processing
6. Unit test and debug
7. System test and debug
8. Installation
9. Maintenance

Although we all agree that the cost of software development exceeds the cost of hardware, we cannot find any common agreement as to particular areas of concentration of cost within this breakdown of the software development process.

CURRENT ACTIVE DEVELOPMENT AND STANDARDIZATION ACTIVITIES

A number of industry-wide and university related standardization and development activities are taking place which will influence trends for process control programming. Among these are the Purdue Workshop on Standardization of Industrial Computer Languages,[49] the American National Standards Institute's PL/I Standardization and Development effort, and several university programs in industrial automation.

Purdue workshop

The Purdue Workshop on Standardization of industrial computer languages has stated as its objectives:

1. To make procurement, programming, installation and maintenance of industrial computer systems more efficient and economical through the development and standardization of industrial computer languages.
2. To minimize educational and training requirements for the use of industrial computer languages.
3. To promote program interchangeability of application programs between different computer systems.

The standing committees of the workshop include:

1. Long Term Procedural Language Committee
2. Problem Oriented Language Committee
3. Glossary Committee
4. Fortran Committee
5. Steering Committee

The workshop met first in the spring of 1969 and approximately every six months thereafter.

AMERICAN NATIONAL STANDARDS INSTITUTE SUB-COMMITTEE X3J1

X3J1 is a sub-committee of Committee X3 of the American National Standards Institute whose scope is the proposal of a draft American National/ISO Standard Composite Programming Language based on PL/I. X3J1 is organized into four working subcommittees which are responsible for:

1. Standardization
2. Development
3. Subsets
4. Development for Industrial Computers

The scope of Committee X3J1.4, PL/I Development for Industrial Computers, is the development of a version of PL/I suitable for application to process control and manufacturing automation by defining and fulfilling the functional requirements of this application area. A basic rule governing this activity is that any string of characters which has validity in the language being developed and the PL/I that X3J1 proposes as a standard must have the same semantics.

X3J1.4 and the Long Term Procedural Language Committee of the Purdue Workshop are currently cooperating to determine if the production of a single result is feasible.

UNIVERSITY AND OTHER ACTIVITIES

Various university laboratories both within the United States and abroad are active in the development of real-time languages and/or programming techniques for industrial computers.

Some of the germinal early real-time language work in the United States was done at Case-Western Reserve University in their Systems Research Center. The Purdue Laboratory for Applied Industrial control continues as sponsor of the Purdue Workshop. Results of significant interest have appeared from the University of Toronto, and German, French and British working groups.

FUTURE TRENDS IN SOFTWARE DEVELOPMENT FOR REAL-TIME INDUSTRIAL AUTOMATION

We have noted above a number of forces which are currently at work to influence future trends:

1. Microelectronics—a capability to produce very powerful central processors at reduced cost.
2. A complex software development problem for real-time application.
3. A current proliferation of programming techniques which are more or less satisfactory, depending upon their match to the application for which they are used.
4. A number of currently active development and standardization activities working in areas which range from customized problem-oriented languages to development of real-time version of PL/I.

Consideration of these forces leads us to believe that the following trends will become evident in the future.

1. Computers with architectures more oriented to the use of special purpose real-time programming languages.
2. Hidden computers.
3. High-level real-time languages.
4. Problem-oriented languages.
5. Software factories combining sophisticated languages and debugging tools for the more effective and efficient production of software on host computers.

New real-time computer architectures

Because real-time computers are special purpose rather than "general purpose" machines, a trend will occur toward more customized architectures, particularly from a programming point of view. Characteristics of these new architectures are likely to include:

1. Bit addressability
   Arrangement of data in a real-time computer in quanta of 8 bits is not very meaningful in the real-time environment. Character oriented data is manipulated relatively seldom in a real-time environment. Convenient methods of accessing data fields of varying widths will be required for convenient implementation of data structures, for example.
2. Direct implementation of real-time features
   Real-time features such as semaphores, more powerful interrupt structures with less inertia, and direct hardware support of event data will ultimately be implemented directly in the hardware.
3. Control structures
   Because sequencing problems represent a high percentage of real time calculations, the instruction repertoire of real-time computers will become more oriented to the easy implementation of more sophisticated control structures.
4. More sophisticated addressing techniques
   Although recursion is not high on the list of requirements for real-time applications, the ability to handle reentrancy in a concise manner is important. This and other requirements such as the manipulation of inhomogeneous data aggregates will dictate more sophisticated addressing modes.

When these more powerful architectures will occur is not entirely evident at this time. It seems that an orderly approach would be to pursue a more consistent definition and understanding of the true real-time language requirements and then let these architectures be designed to support such languages (Refs. 50 through 60).

Hidden computers

The capability to build special purpose hardware will, in the future, increase the trend toward distributed digital systems. The computational capability of real-time process control systems will not be concentrated in one central processor as is typical now but will appear both in the central processor and a number of remote dedicated processors which function to gather intelligence about the process under control, execute direct control functions and communicate to the central processor for supervisory control functions. The major force which will cause this to occur is what is sometimes called "the cost of copper." This expression refers to the fact that in many installations it is possible to spend as much installing the wiring to carry signals from sensors and control devices back to the computer as on the computer itself. The impact of this trend upon programming techniques is that these devices will, to all intents and purposes, be hidden from the user. They will perform their functions in a manner analogous to hardwired hardware devices in current use, and the programming problems associated with direct control will frequently be segregated from the rest of the system. As an example of the importance of this, it has been found that in the development of large systems for both supervisory and direct digital control that such major differences in operating systems are required to support these two control techniques as to make organization of a system with a combination of them rather difficult and costly. The use of hidden computers will reduce this problem significantly.

Problem oriented languages

For those users who have applications of some generality, problem oriented languages and systems seem to be unquestionably the best long term solution to their software development problems. There is no question but that a truly general package incurs a penalty in overhead and excess hardware requirements, but with the cost trends in hardware which we have discussed, it will become more and more apparent that this is the most economical solution. As an example, our experiences with certain problem oriented packages have been that they can reduce the programming requirements by a factor of 10 and allow personnel whose major area of specialty is the application rather than software to do the programming. This usually results in a much better control system. Thus, on the belief that what is best for the user will eventually ensue, we predict that for many of them, their problems will be solved by problem oriented systems which will very much minimize their requirements for a professional programming staff.

Real-time procedural languages

As we have seen, the major real-time languages in current use are variants of Fortran. Table I shows some of the typical extensions which have been made in Fortran for this application area. It is clear from examining the length and breadth of these extensions that if an all-encompassing procedural language based on Fortran were to be implemented, the base language would disappear. For this reason, current development activities are now turning in a different direction.

The general goals of real-time procedural language development activities under way today are to develop a language which leads to:

1. Lucidity—the clear expression of what is to occur when a program is executed, providing a higher level of self documentation.
2. Freedom from side effects—in a real-time environment a predictable system behavior is extremely important and real-time programming languages must be free from side effects.
3. Enforcement of programming discipline—in the business of producing software, it is clear that programming discipline is increasingly important. A well designed language will help maintain this discipline.
4. Natural modularity—the language must be constructed so as to make easy increased modularization of real-time systems.
5. Ease of learning and use—it must not require a major investment in training to put a new language into use, if possible.

Also of particular interest to the designer of a real-time procedural language is the maintenance of a proper balance between run time efficiency, integrity

TABLE I—Typical FORTRAN Extensions

- 1 Program & Data Structure
  - (a) Data types
    - Fixed point
    - Bit
    - Status
    - Byte
    - Binary
    - Boolean (vector)
    - Octal
    - Hexadecimal
    - Alphanumeric
  - (b) Data manipulation
    - Shifts
    - Logical operators
    - Bit and byte replacement
    - Character manipulation
    - String manipulation
    - Bit testing
- 2 Communications
  - (a) Between programs
    - Global common
    - External data
    - Static file system
    - Dynamic file system
  - (b) Input/Output
    - Unformatted input/output
    - Read write bulk
- 3 Program control
  - Schedule another program
  - Link with interrupt
  - Delay
  - Look at clock
  - Terminate execution
  - Decision tables
- 4 Compiler features
  - Diagnostic trace
  - Conditional compilation
  - Reserved words
  - Code optimization directives

TABLE II—Some Functional Requirements for a Real-Time Language

I. *INTERRUPT HANDLING & SYNCHRONIZATION*

- A. Interrupts
  1. Ability to indicate that some program segment is to be executed upon the occurrence of some particular interrupt.
- B. Events
  1. Ability to retain the flow of control at a point in a computational process until some event has occurred.
  2. Ability to cause the occurrence of an event at some point in a computational process.
- C. Synchronization
  1. Ability to synchronize parallel computational processes with security.

II. *TASKING*

- A. By a task on itself.
  1. Schedule Execution
  2. Find Status
  3. Exit
  4. Kill
  5. Delay
  6. Wait
- B. On another task
  1. Execute
  2. Schedule
  3. Suspend
  4. Delay
  5. Terminate
  6. Set Priority
  7. Find Status

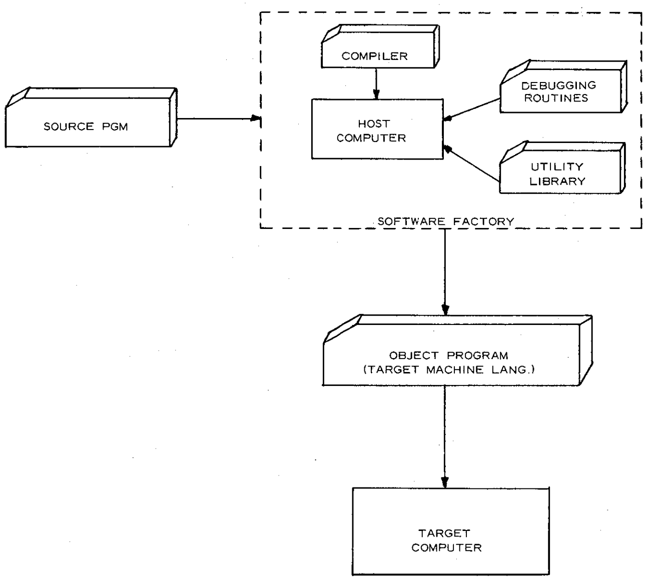


Figure 2—Software factory

of the system and ease of programming. A general set of functional requirements for a real-time language is shown in Table II. Whether these languages will be general purpose[61] or dedicated to particular architectures[62] remains to be seen.

Software factories

Just as the ultimate elimination of the requirements do a great deal of custom programming by the use of special purpose application packages is the ultimate answer in reducing software costs for the users of real-time automation systems, the development of a unified, well-designed software will provide the mechanisms for more economical development of such systems. Figure 2 shows the general nature of such a software factory. It consists of four major elements; a compiler for a high-level language for systems software development, a utility library of building block software modules, a well organized approach to debugging software produced in the software factory, and a powerful host computer upon which the system is run.

The general-purpose real-time language described above will be used with the debugging system to develop the utility library and various application packages.

Debugging system

One of the major motivations for using a large host computer for software production is the more powerful facilities which it makes available for software debugging. The well designed debugging system for the software factory will attack three problem areas:

1. Observability—to make more visible the actions of the program being tested.
2. Controllability—to allow the programmer to control the flow of control within his program to suspect control paths.
3. Repeatability—to insure that the same response occurs from given stimuli, aiding problem diagnosis.

Table III lists the features generally used in such a system to achieve these goals.

There is still some disagreement between the use of source language debugging through compilation to host machine executable programs, and direct instruction simulation of the target computer. Each point of view has merit. The source language execution approach allows speedier execution and provides for quick elimination of gross structural errors in the program. Simulation of the target computer at the instruction level is generally more expensive because of increased execution requirements on the host computer, but allows one to locate more subtle errors, particularly those with respect to behavior of the target machine hardware in unusual timing circumstances. (The latter requires, of course, an extremely sophisticated simulation of the target computer system.) In the future both techniques will play an important role (References 63-68). (It is interesting to note, however, that the ultimate distinction may be moot as target machines come closer and closer to executing high level languages directly.)

Utility library development

Perhaps the most significant aspect of this overall software factory will be the availability of software modules for use in building real-time systems. The

TABLE III—Features Required in Debugging Systems

1. Breakpoint removal and insertion
2. Online variable revision
3. Insertion and removal of snapshot dumps
4. Execution initiation or transfer of control to any point
5. Fault trapping
6. Incremental addition of new instruction
7. Variable searches by value
8. Subroutine call and operating system linkage tracing
9. Input/output and interrupt simulation
10. Control of internal timing

modules required may be grouped in four categories:

1. System building modules
   These modules will form the components of various language processing systems desired by the user. With the use of these modules, the development of special purpose language processors for various applications areas will be made much more economical.
2. Application modules
   Modules of on-line application software will be available for combination into application-oriented control software packages. Examples of these modules will be software for direct digital control, supervisory control, and real-time data base management.
3. Systems operating modules
   Operating systems as we know them today may very well disappear in the future. Instead, the various operating system components like interrupt handlers, generalized input/output systems, priority schedulers and executives will be available for combination as required.
4. User interface modules
   Customized user interface modules will be available for each application area. These modules will actually include the linkages necessary to combine the library modules and any specially developed software into a functioning control system.

SUMMARY AND DISCUSSION

We have examined a number of interesting trends in real-time software development. Of particular interest is the three-way interaction foreseen between the development of new, more powerful architectures for real-time computers, more powerful high-level programming languages, and the integration of these programming languages into a more efficient software production system.

The most interesting technical challenge in this situation is the opportunity to develop languages and architectures in a unified manner. In the past the designer of computer architectures has been far removed from consideration of the application requirements, and his products have been extremely difficult to apply. The designer of real-time programming languages now has the opportunity to act as a coordinator in this overall process while insuring that the language fulfills the functional requirements of the application area hand, he will coordinate with the compiler developer and computer architecture designer to insure that the resultant combination of languages, software factory, and computer form an integrated system for the efficient solution of real-time automation problems.

Exact timing of these trends is very hard to forecast. Working prototypes of most of the items discussed exist today. When the economics of the marketplace are satisfactory, these prototypes will be turned into smoothly functioning systems for general use. Development of languages and architectures are already under way. As this work matures the software factory concept will provide the unifying factor which knits this all together.

REFERENCES

1 ______
L6P-30 Royal Precision Electronic digital computer operations manual
Royal McBee Corp 1959

2 J C SCHOEFFLER
The development of process control software
Proceedings of the 1972 Spring Joint Computer Conference

3 ______
Proceedings of the IEEE-special issue on computers in industrial process control
Vol 58 No 1 January 1970

4 C L SMITH
Digital control of industrial processes computing survey
Vol 2 No 3 September 1970

5 A S BROWER
Digital control computers for the metals industry
ISA J Vol 12 pp 51-63 February 1965

6 B O JENDER
Problems in computer control of bleach plants
Paper Trade J pp 48-50 May 26 1969

7 D S HOAG
Computer control of a Kamyr digester
Presented at the 2nd Ann Workshop on the Use of Digital Computers in Process Control Louisiana State University Baton Rouge La March 1967

8 M R HURLBUT et al
Applications of digital computer control to the cement manufacturing process
International Seminar on Autom Control in Lime Cement and Connected Industries Brussels Belgium September 1968

9 T H LEE G E ADAMS W M GAINES
Computer process control: Modeling and optimization
Wiley New York 1968

10 T M STOUT
Where are process computers justified?
Presented at the International Symposium on Pulp and Paper Process Control Vancouver Canada April 1969

11 T M STOUT
Process control
Datamation 12 2 Feb 1966 pp 22-27

12 T J WILLIAMS
Economics and the future of process control
Automatica 3 1 October 1965 pp 1-13

13 T J WILLIAMS
Computer systems for industrial process control, A review of progress, needs and expected developments
Proc 1968 IFIP Congress

14 N COHN et al
On-line computer applications in the electric power industry
Proceedings of the IEEE Jan 1970 Vol 58 No 1 pp 78-87

15 _____
Guidelines and General Information on User Requirements Concerning Direct Digital Control
Control Instrument Society of America Pittsburgh 1969 pp 251-258 and 259-278

16 H E PIKE
Direct digital control—A survey
23rd Annual ISA Conf and Exhibit New York October 1968 Preprint 68-840

17 T J WILLIAMS E M RYAN Eds
Progress in direct digital control
Instrument Society of America Pittsburgh Pa 1969

18 H E PIKE
Process control software
Proceedings of the IEEE Jan 1970

19 H E PIKE
Real-time software for industrial control
In International Computer State of the Art Report Real-Time Infotech Ltd Maidenhead Berkshire England 1971

20 R E HOHMEYER
CDC 1700 Fortran for process control
IEEE Trans Indus Electronics & Cont Instmn IECI-15 No 2 p 67 1968

21 D R FROST
FORTRAN for process control
Instr Technol April 1969

22 J C MECKLENBURGH P A MAY
Protran, a fortran based computer language for process control
Automatica 6 No 4 p 565 1970

23 M MENSCH M DIEHL
Extended fortran for process control
IEEE Trans Indus Electronics and Control Instrm IECI-15 No 2 p 75 1968

24 M MENSCH W DIEHL
Extended FORTRAN for process control
Presented at the Joint Automatic Control Conf 1968

25 W DIEHL
An integrated hardware/software approach for process control
Presented at the ISA Conf New York New York 1968

26 W DIEHL M MENSCH
Programming industrial control systems in fortran
IFAP/IFIP Symposium Toronto 1968

27 B C ROBERTS
FORTRAN IV in a process control environment
Presented at the Joint Automatic Control Conf 1968

28 _____
Special issue on computer languages for process control
IEEE Transactions on Industrial Electrons and Control Instrumentation IECI-15 Dec 1969

29 G W MARKHAM
Fill-in-the-form Programming
Control Engineering May 1968

30 _____
Digital Equipment Corporation Indac 8 DEC Maynard Mass 1969

31 T G GASPAR V V DORBROHOTOFF
D R BURGESS
New process language uses English terms
Control Eng 15 No 10 p 118 1968

32 _____
AUTRAN training manual
Control Data Corp La Jolla Calif 1968

33 _____
1800 process supervisory program (PROSPRO/1800) (1800-CC-02X) language specifications manual
IBM Corp No H20-0473-1 1968

34 S M FISCH S J WHITMAN
An application of a process control language to industrial digital control
Presented at the ISA Ann Conf New York NY 1968

35 _____
Control Data 1700 AUTRAN process control computer system
Control Data Corp La Jolla Calif 1968

36 _____
BICEPS summary manual/BICEPS supervisory control
GE Process Computer Dept Phoenix Ariz A GET-3539 1969

37 _____
BATCH sequencing system
Foxboro Co Foxboro Mass TIM-R-97400A-504 1968

38 J BRANDES et al
The concept of a process-and-experiment-orientated programming language
Electronishe Datenverarbeitung 10 429 1970

39 P A MAY J C MECKLENBURGH
Protol, an Algol based programming language for process control
Chem Eng J 2 No 3 p 60 1971

40 R TEMPLE J D SCHOEFFLER
A real-time language for computer control software
Systems Research Center Report Case Western Reserve University Cleveland Ohio 1969

41 R TEMPLE
RTL—A real-time language for computer control software
PhD Thesis Case Western Reserve University Cleveland Ohio 1969

42 J D SCHOEFFLER T L WILLMOTT
J DEDOUREK
Programming languages for industrial process control
IFAC/IFIP Congress Menton France 1967

43 T L WILLMOTT
A programming language for process control computers
MS Thesis Case Institute of Technology Cleveland Ohio 1967

44 P A MAY
Development of software for real-time computer systems
PhD Thesis University of Nottingham UK 1971

45 J GERTLER
Some concepts on the programming of direct digital process control systems
Acki Technica Hung 61 No (1-2) p 55 1968

46 J GERTLER
High level programming for process control
Comp J 13 No 1 p 70 1970

47 BCS Specialist Group
A language for real-time systems
Comp Bulletin No 3 p 202 1967

48 _____
TI calculator chip bids to make a wave
Electronics Sept 27 1971 p 24

49 _____
Workshop on standardization of industrial computer languages
Purdue University Lafayette Indiana Feb Sept 1969 March Nov 1970 May 1971

50 L CONSTANTINE
Integral hardware/software design
A multi-part series that appeared in Modern Data from April 68 through February 69

51 F S KEELER et al
Computer architecture study
SAMSO report TR-70-420 available from NTIS as AD720798

52 C C CHURCH
Computer instruction repertoire—Time for a change
Proceedings of the 1970 Spring Joint Computer Conference

53 H WEBER
A microprogrammed implementation of Euler on the IBM 360/30
CACM 10 1967 549-558

54 _____
B6500 information processing system reference manual
Burroughs Corporation Manual #1043676

55 R RICE W R SMITH
SYMBOL—A major departure from classic software dominated von Neumann computing system
Proceedings of the SJCC 1971 3, 8 AFIPS Press pp 575-587

56 C McFARLAND
A language-oriented computer design
Proceedings of the FJCC 1970 37 AFIPS Press pp 629-640

57 P S ABRAMS
An APL machine
Stanford Electronics Laboratory—Technical Report No 3 Feb 1970

58 T F SIGNISKI
Design of an algol machine
Computer Science Center Report 70-131 University of Maryland Sept 1970

59 G W PATTERSON et al
Interactions of computer language and machine design
Moore School of Electrical Engineering Report No 63-09 University of Pennsylvania Oct 1962

60 R W DORAN
Machine organization for algorithmic languages
Proceedings of the International Computing Symposium Bonn Germany May 1970 pp 364-376

61 F J CORBATO
PL/I as a tool for system programming
Datamation 15 1969 pp 68-76

62 N WIRTH
PL/360—A programming language for the 360 computers
Journal of the Association for Computing Machinery Vol 15 #1 Jan 1968

63 D T ROSS
The AED approach to generalized computer-aided design
Proceedings of the 1967 ACM National Meeting pp 367-385

64 D T ROSS
Fourth generation software: A building-block science replaces hand-crafted art
Computer Decisions April 1970

65 R M SUPNIK
Debugging under simulation
Applied Data Research Inc Princeton NJ

66 _____
MIMIC user's guide—PDP-11
Manual CSD 03-70411M-00 Applied Data Research Inc Princeton NJ

67 M L GREENBERG
DDT: Interactive machine language debugging system reference manual
NTIS Report AD707406

68 W KOCHER
A survey of current debugging concepts
NASA report CR-1397 available from NTIS as N69-35613

Scheduling of time critical processes

by OMRI SERLIN

SYSTEMS Engineering Laboratories
Fort Lauderdale, Florida

INTRODUCTION

In real-time applications, the computer is often required to service programs in response to external signals, and to guarantee that each such program is completely processed within a specified interval following the occurrence of the initiating signal. Such programs are referred to in this paper as time-critical processes, or TCPs.

A problem of considerable practical significance in connection with such applications is that of devising scheduling algorithms for servicing multiple time-critical processes. The primary objective of any such algorithm is to guarantee that each TCP can be processed within its allowed interval, taking into account the possible activities of other TCPs. It is clear that, assuming an upper limit can be placed on the repetition rate of the prompting signals, this objective can always be met if, for example, each TCP executes in a separate, sufficiently fast CPU; or, if a single CPU of some high, but finite, execution speed is furnished. Either approach is wasteful in the sense that the CPU (or CPU's) must sometimes remain idle in order to allow for some "worst-case" condition that occasionally may obtain; for example, when all interrupts occur simultaneously. Thus the scheduling problem becomes non-trivial only if additional constraints are imposed. Two such practical constraints are that only one CPU is available, and that the processor idle time is to be minimized.

This paper examines four specific scheduling policies, subject to these constraints, and outlines some of the more interesting properties of each.

DEFINITIONS

Time critical processes

In the context of computing systems, time critical scheduling is the problem of devising *efficient* CPU allocation policies that satisfy the requirements of multiple *time-critical processes* (TCPs). These processes, arising naturally in real-time environments, are computational procedures bound by *hard deadlines.* A process bound by a deadline must, once initiated, complete its computation on or before the specified deadline. A *hard* deadline implies that a failure to meet it results in an irreparable damage to the computation. Such deadlines are typically due to stability and accuracy considerations in the process being controlled (e.g., the analog computation portion of a hybrid system).

An efficient CPU allocation algorithm is one that guarantees to each TCP sufficient processor time to complete its task before its deadline, while minimizing *forced* idle CPU time. By "forced idle time" is meant the time during which the CPU must remain idle in order to accommodate the occasional worst-case condition. This constraint is essential if the problem is to be non-trivial.

A basic assumption underlying this study is that all TCPs are preemptible.

Time-critical processes are most commonly initiated by the occurrence of external events (*interrupt signals*). Time-of-day or elapsed-time prompting of these processes may be regarded as special cases of the general class of external events. These events, and the associated TCPs, may be further classified as *periodic* or *asynchronous.*

The TCPs to be serviced are assumed to form an unordered set, that is, they are not interrelated by precedence rules. This assumption is consistent with the premise that the TCPs are initiated by unrelated external signals. If a TCP calls on other tasks, then such secondary tasks can be regarded simply as chronological extensions of the calling TCP, either because the called task is private to the calling TCP, or because all shared tasks are reentrant. In fact, the concept of precedence rules is relevant only in the case of scheduling for multiprocessor systems (Ref. 3). Multiprocessor configurations, however, seem to offer no advantages in

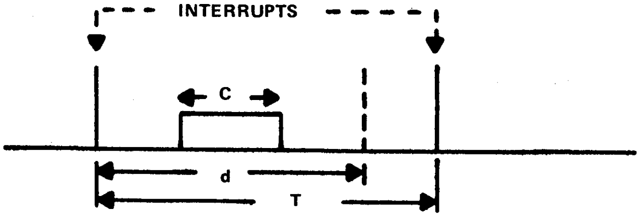


Figure 1—A model of a time critical process (TCP). The parameters associated with the TCP are the deadline (d), repetition period (T), and required CPU time (C)

the implementation of time critical schedules of the type discussed here. This subject is discussed later.

Note, incidentally, that the grouping of interrupt-driven processes into entities called *jobs* is primarily a bookkeeping measure. Such groupings carry certain implications as to the sequence of events before and after the time critical phase of the operation; they are of interest to the loader and accounting mechanisms. From the scheduling viewpoint, however, TCPs are independent entities; it does not matter whether all belong to a single user ("job") or whether sets of one or more TCPs each belong to one of several users.

TCP models

Figure 1 is a model of a simple TCP, depicting its repetition period (T), deadline (d), and required compute time (C). The required compute time is the maximum time required to completely process the given TCP on a given CPU, assuming no interruptions. That is, C is the amount of CPU service, in terms of CPU-seconds, required by the TCP. Evidently, C depends on the number and type of executable instructions in the TCP and the speed of the processor. The required compute time may be obtained empirically by timing the process running alone, or it could be predicted by deriving, through such procedure as the Gibson mix, an average instruction execution time for the processor and multiplying this figure by the instruction count of the given process.

A basic assumption underlying the study of TCP scheduling is that the external world is insensitive to the actual time at which the process is executed, so long as the process receives C CPU-seconds somewhere between the occurrence of the associated interrupt and the deadline. This is typically achieved by incorporating in the process predictive algorithms that extrapolate the results computed from data obtained at time t_i to t_i+d, when these results are actually transferred out of the computer.

T is the repetition period, i.e., the period between successive occurrences of the interrupt signal associated with the process. In practice, TCPs often are periodic and have $d=T=$constant; i.e., the constant repetition period and the deadline are identical. The repetition period is often called the frame time. Typical asynchronous TCPs have $d<T$, and T is variable. Nevertheless, these processes may still be represented by the model of Figure 1, as the following considerations show. It must be possible to define for an asynchronous TCP a *minimum* T; for if this cannot be done, then either the process cannot be time-critical—namely, it cannot be bound by a hard deadline—or else a processor of infinite execution speed is required. Hence, in the discussions to follow, T is taken to mean either the constant repetition period of a periodic TCP, or the minimum time between interrupts for an asynchronous process.

It is generally necessary to assume that TCPs can become active, either periodically or asynchronously, at unpredictable times. Furthermore, it is necessary to assume that no special relationships exist between or may be imposed on the frame times and compute times of the various processes. In practical situations, however, the resolution of the system's timing mechanisms may dictate that all Ts and Cs be integer multiples of some basic time unit, so that all Cs and Ts are *mutually commensurable*. This property can be quite significant in devising realistic schedulers.

As a rule, time-critical processes require the input of external data before they may proceed, and must allow some time after their completion and before the occurrence of their next interrupt for computed date to be output to external devices. Figure 2 is a model of a TCP of this type. A simple re-definition of the repetition period and deadline (T' and d' in Figure 2) transforms this type to the basic model of Figure 1. This transformation notwithstanding, consideration of I/O requirements complicate the scheduling problem, as will be pointed out later. It is assumed that, apart from an initial input and a final output operations, TCPs do not perform I/O.

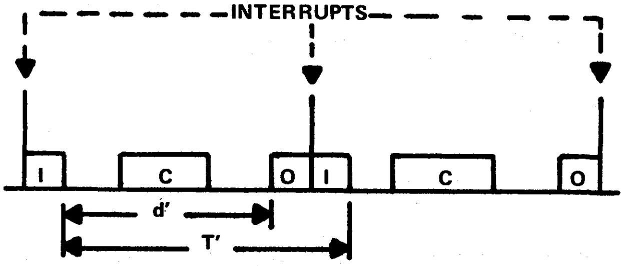


Figure 2—A TCP with I/O. By transforming the deadline into d' and the repetition period into T' this TCP is made to conform to the basic TCP shown in Figure 1

Load factor

The *load factor* (L) of a given process is equal to C/T, the ratio of the required CPU time to the repetition period. The load factor is a measure of the CPU load represented by the given TCP. Because the load factor can be readily defined in terms of quantities that are private to a given TCP and are easily obtained, it is natural to seek to establish a clear relationship between the individual load factors and the total CPU load. For instance, it is clear that in the global sense—i.e., over a very long period compared to all the frame times—the sum of the load factors approximates the total CPU load; hence, the condition $\sum L > 1$ is sufficient to guarantee the failure of all scheduling policies. On the other hand, the condition $\sum L \leq 1$ is *not* sufficient to guarantee success, as will be demonstrated shortly. An efficient scheduler will maximize the sum of the load factors without violating any deadlines.

Overhead

Overhead is the time required to switch the processor state from one TCP to another. In the following discussions, overhead is assumed to be zero. For that reason a more detailed definition of overhead, which takes into account the variety of possible hardware and software configurations affecting it, is unnecessary. In many practical cases the assumption of zero overhead is a good first approximation, since the state switching time is indeed small compared to the frame times.

THE INTELLIGENT FIXED PRIORITY (IFP) ALGORITHM

In the most commonly used scheduling technique, each TCP is assigned a fixed priority at the time it enters its time-critical phase—that is, when its associated interrupt is armed and enabled. If the priority levels are such that *processes having shorter deadline intervals receive higher priorities*, the algorithm is termed "intelligent" fixed priority, to distinguish it from other fixed-priority systems in which the allocation of priority levels is based on other criteria. This should not be taken to mean that the choice of such other criteria is unintelligent; for example, a scheduling policy based on *no* priorities ("round robin"), or one in which "CPU hogs" receive low priorities, can be quite effective in time sharing environments or in multiprogrammed batch systems. For the time-critical environment relevant to this discussion, the allocation of priorities based on the relative lengths of deadlines is most appropriate.

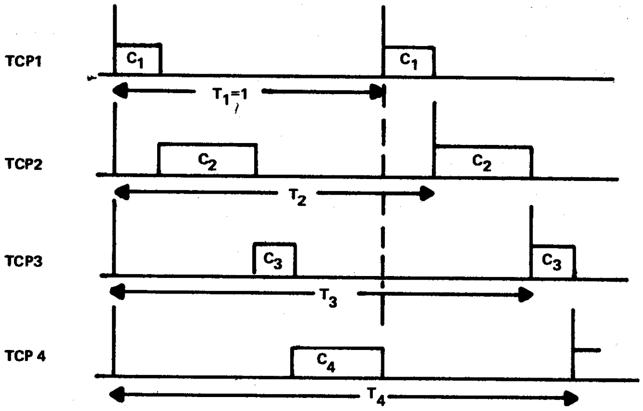


Figure 3—Assumed worst case under IFP. Four TCPs are shown, satisfying the conditions that are considered "worst-case": (a) all interrupts occur simultaneously; (b) the frame times are so related that the CPU must process all TCPs completely within the shortest frame time

Priorities under IFP need not be truly fixed. In a multiprogrammed real-time system, groups of TCPs ("jobs") may enter and leave the system at arbitrary times; the priority structure must be adjusted occasionally in response to such changes in the relative deadlines. Also, real-time processes often go through pre- and post-time-critical phases, during which their priorities may be much lower than the ones they assume in their time-critical phase, again suggesting a dynamic readjustment.

An important property of IFP, which has been alluded to in References 1 and 2 but heretofore not substantiated, is that the permissible sum of the load factors must be considerably less than 1 in order to guarantee that each TCP meets its deadline. The higher the number of TCPs to be scheduled, the closer this permissible sum gets to $\log_e 2 = 0.693$. In other words, under IFP, more than 30 percent of the CPU power must remain idle. The following is a sketch of the proof.

Assume there are n TCPs, all with constant frame times, and that, at one instant, all of their interrupts occur simultaneously. The CPU at that time is required to provide maximum service in minimum time, so that this condition represents the worst possible case. Furthermore, assume that the frame time of each TCP is related to the next shorter one by $T_i = T_{i-1} + C_{i-1}$ where $T_{i-1} < T_i < 2T_{i-1}$ and $d_i = T_i$ or $d_i = \sum_{j=1}^{i} C_j$ (See Figure 3).

These conditions assure that the CPU will be 100 percent busy during the longest frame time Tn following the simultaneous occurrence of the interrupt signals. We wish to establish the set of T_i, C_i that lead to the smallest possible sum of the load factors, i.e., the largest possible forced CPU idle time.

Normalize the frame times by letting $T_1=1$. Designate the sum of the load factors by z. Then

$$z=C_1+C_2/(1+C_1)+C_3/(1+C_1+C_2)+\cdots+C_n/(1+C_1+C_2+\cdots+C_{n-1})$$

Because of the assumptions (see Figure 3),

$$\sum_{j=1}^{n} C_j=1$$

Let

$$f=z+L\left(\sum_{j=1}^{n} C_j-1\right)$$

where L is a Lagrange multiplier used for convenience to obtain symmetry in the following equations. Also let

$$1+\sum_{j=1}^{n} C_j=S_i.$$

We wish to find the conditions leading to the minimum z. Hence:

$$\begin{aligned}
\partial f/\partial C_1 &= 1-C_2/S_1^2-C_3/S_2^2-\cdots-Cn/S_{n-1}^2+L=0\\
\partial f/\partial C_2 &= 1/S_1-C_3/S_2^2-\cdots-Cn/S_{n-1}^2+L=0\\
\partial f/\partial C_3 &= 1/S_2-\cdots-Cn/S_{n-1}^2+L=0\\
&\vdots\\
\partial f/\partial C_{n-1} &= 1/S_{n-1}-Cn/S_{n-1}+L=0\\
\partial f/\partial C_n &= 1/S_{n-1}+L=0
\end{aligned}$$

By subtracting each of these equations from the one above it we obtain

$$\begin{aligned}
S_n &= S_{n-1}^2/S_{n-2}\\
S_{n-1} &= S_{n-2}^2/S_{n-3}\\
&\vdots\\
1+C_1+C_2 &= (1+C_1)^2
\end{aligned}$$

where the last equation is obtained from

$$\partial f/\partial C_1-\partial f/\partial C_2=1-C_2/(1+C_1)^2-1/(1+C_1)=0$$

Continual upward substitution in the above set leads to

$$1+C_1+C_2+\cdots+C_n=(1+C)^{2(n-1)}/(1+C)^{n-2}$$

The left hand side of the equation above equals 2 since $S_n=1$; hence

$$2=(1+C_1)^n$$

so that

$$C_1=2^{1/n}-1$$

This represents both the required compute time and load factor for TCP 1 under the "worst case" condition of Figure 3 with the side constraint of worst (minimum) load-factor sum. The other TCPs have

$$C_i=S_{i-1}/S_{i-2}^2-S_{i-1}$$

By continual substitution, we have

$$S_i=(1+C_1)^i$$

The load factor L_i of each TCP$_i$ is C_i/T_i where $T=S_{i-1}$; so

$$L_i=[(S_{i-1}^2/S_{i-2})-S_{i-1}]/S_{i-1}=C_1$$

so all the load factors equal C_1. Thus the sum of the load factors is

$$z=n(2^{1/n}-1)$$

Note, incidentally, that the frame times for this worst-case core given by

$$T_i=S_{i-1}=(1+C_1)^{i-1}=2^{(i-1)/n}$$

The table below summarizes the worst-case situation for two, three and four TCPs.

n	z (max. L)	T relationships
2	$2(\sqrt{2}-1)\cong$82.8 percent	$1:\sqrt{2}$
3	$3(\sqrt[3]{2}-1)\cong$78.0 percent	$1:\sqrt[3]{2}:\sqrt[3]{4}$
4	$4(\sqrt[4]{2}-1)\cong$75.7 percent	$1:\sqrt[4]{2}:\sqrt[4]{4}:\sqrt[4]{8}$

To establish the trend as the number of TCPs increases, we take the limit of z as n approaches infinity:

$$\begin{aligned}
\lim_{n\to\infty} n(2^{1/n}-1) &= \lim_{n\to\infty} n[1+\ln 2/n+(\ln 2/n)^2/2!\\
&\quad+(\ln 2/n)^3/3!+\cdots-1]\\
&= \log_e 2=.69315
\end{aligned}$$

thereby substantiating the original assertion. Figure 4

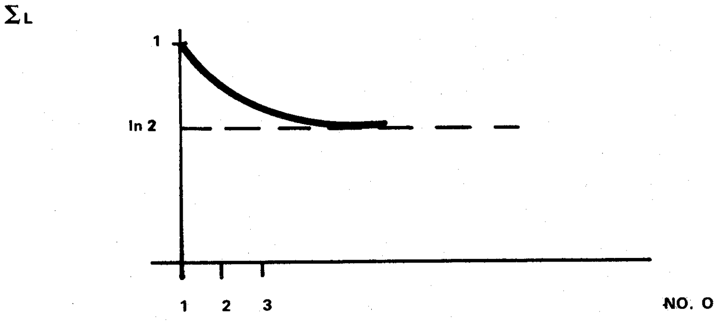


Figure 4—Load factor sum as a function of the number of TCPs under IFP. As the number of TCPs increases, the CPU must remain idle a larger percent of the time (i.e., the permissible sum of the load factors must be reduced). In the case of many TCPs, about 30 percent of the total CPU power must be held in reserve to accommodate the worst-case condition

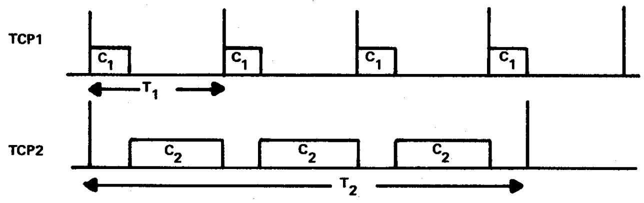


Figure 5—Worst case situation for two TCPs under IFP when the condition $T_1 < T_2 < 2T_1$ does not apply

shows the maximum permissible load factor sum as a function of the number of TCPs.

It is of interest to investigate the case where the frame times are not restricted to the range $T_{i-1} < T_i < 2T_{i-1}$. Figure 5 shows the effect of removing this restriction for the case $n=2$. For the situation shown in Figure 5 we have:

$$T_2 = kT_1 = mT_1 + C_1$$

where $m > 1$ and is an integer. Also,

$$C_2 = m(T_1 - C_1)$$

hence the sum of the load factors is

$$z = C_1/T_1 + m(T_1 - C_1)/kT_1$$
$$= k - m + m(1 - k + m)/k$$

For a minimum z we require

$$\partial z/\partial k = 1 - (m^2 + m)/k^2 = 0$$

Hence,

$$k = \sqrt{m(m+1)}$$

describes the points at which the ratio T_2/T_1 minimizes the available CPU time. At these points, the load factors are

$$C_1/T_1 = \sqrt{m(m+1)} - m = C_2/kT_2$$

and the sum of the load factors is

$$z = 2(\sqrt{m(m+1)} - m)$$

By assuming a given ratio of T_1 to T_2 with the "worst case" constraints $T_2 = mT_1 + C_1$ and $C_2 = m(T_1 - C_1)$ it can be shown that the sum of the load factors as a function of T_2 (or C_1) is given by

$$z = C_1 + m(1 - C_1)/(m + C_1)$$

Using these last two results for z it is possible to develop Figure 6 which shows how the maximum permissible sum of the load factors varies as a function of the ratio T_2/T_1. When this ratio is an integer, there is no problem in allocating 100 percent of the CPU using IFP. When T_2/T_1 is not an integer, some CPU time must remain idle in order to accommodate the "worst case." It is interesting to note that the CPU utilization drops to its minima always when T_1 and T_2 are *mutually incommensurate.*

EFFECT OF I/O

It was pointed out earlier that TCPs with nonzero I/O times can be transformed to the model of Figure 1. Such a transformation does not, however, remove a basic difficulty associated with I/O: during the input or output phases of a given TCP, the CPU may not be allocated to it. During the time that the I/O operations of all TCPs overlap, the CPU may not be allocated to any. For example, with two TCPs of identical repetition periods T and equal input-output phases I, the maximum permissible sum of the load factors is $(T-2I)/T$. This difficulty may be further complicated by the need to schedule the I/O operations that use the same I/O facilities.

Assuming, however, that all I/O operations can be performed simultaneously, one possible approach to incorporating the effect of I/O in CPU scheduling policies may be as follows. Regard I/O as a TCP with $d_i = T_i = C_i$, which is always executed on a second, "dummy" CPU. Furthermore, introduce precedence rules enforcing the sequence $I_i < \mathrm{TCP}_i < O_i$ (read: task I_i precedes task TCP_i etc.). Unfortunately, these artifices do not simplify significantly the analysis of scheduling problems with non-zero I/O times. In the remainder of this paper, zero I/O times are assumed.

THE INFINITE TIME SLICING (ITS) ALGORITHM

Although this algorithm cannot be implemented, it is nevertheless a valuable tool in the study of time critical

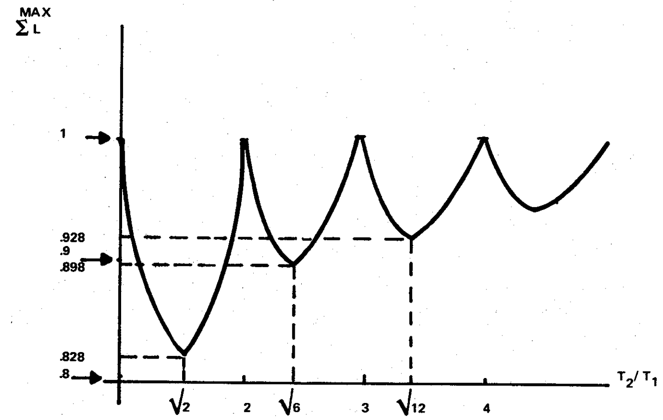


Figure 6—Maximum permissible load factor sum as a function of the frame times ratio T_2/T_1, Two TCPs. The minima occur when T_2 and T_1 are mutually incommensurate

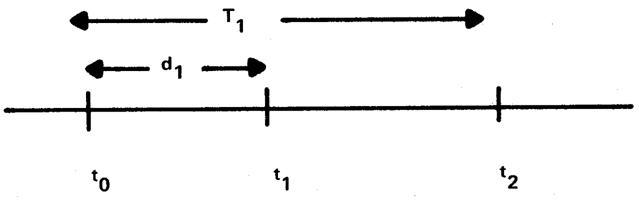


Figure 7—Nomenclature for the proof that 100 percent scheduling is impossible unless $d_i = T_i$ or $T_i = mT_1$

scheduling. ITS provides a conceptual link between the sum of the individual load factors and the instantaneous CPU load. Namely, given a set of TCPs for which the sum of the load factors is L, then, using ITS it is possible to guarantee that (as long as $d_i = T_i$) all deadlines are met while the CPU is busy exactly $100L$ percent of the time in any given time span.

ITS may be approached in the following way. Suppose it were possible to regard the power of the CPU as a continuous quantity; it may then be distributed to the TCPs according to the load factors. In other words, assuming $d_i = T_i$ for all TCPs, then a TCP whose load factor is $L_i < 1$ is allocated a fraction of the CPU equal to L_i. This may be taken to mean that if the CPU is capable of executing G instructions per second, then an L_i fraction of the CPU has the ability to execute L_iG instructions/sec; or, the CPU may be regarded as a composite of a number of slower CPUs, each performing L_iG instructions/sec. If this is done, then each process executes exactly L_iGT_i instructions in its period T_i. Since $L_i = C_i/T_i$, the instruction sum is C_iG, which is exactly the TCP's requirement when assuming C_i cpu seconds at G instructions/sec.

To achieve such a distribution of the CPU power one can start by dividing the time axis into small increments ΔT. In each increment, the CPU is allocated, in some arbitrary order, to each TCP for a period $L_i\Delta T$. If ΔT is made small compared to all the repetition periods T_i involved, we have a good approximation to the infinite-time-slicing algorithm. An "exact" ITS policy is obtained if we let ΔT approach 0. Approximate ITS algorithms can be—and, in fact, have been—implemented on computers that provide rapid context switching. With context-switch time of, say, 5 microseconds, and ΔT set at 250 microseconds, it is possible to service five TCPs with 10 percent switching overhead.

Note however that ITS is weak when required to schedule TCPs for which $d_i < T_i$. For such processes, ITS must allocate C_i/d_i fraction of the CPU. This leads to situations where a specific TCP set that can be successfully scheduled by IFP (and other policies) fails under ITS. For instance, given a periodic process with $d_1 = T_1 = 100$ milliseconds and $C_1 = 85$ milliseconds, and an asynchronous process with minimum $T_2 = 2$ *ms*, $d_2 = 1$ *ms*, and $C_2 = 200$ microseconds. To schedule these, ITS must allocate $85/100 = 85$ percent of the CPU to TCP 1 and $0.2/1 = 20$ percent to TCP 2; hence these TCPs cannot be scheduled together. Under IFP, both TCPs can easily be accommodated if TCP2 is given the higher priority, since TCP2 requires a maximum of 10 *ms* CPU time during the frame time of TCP 1.

A BASIC SCHEDULING RESTRICTION?

At least in the case of two TCPs it appears that the following statement is true. It is never possible to allocate the CPU at full efficiency (i.e., keep it busy doing useful work 100 percent of the time) unless either $d_i = T_i$ or $T_2 = mT_1$, where m is an integer.

For the case $d_i < T_i$, consider Figure 7. Assume the CPU is allocated 100 percent, i.e., no idle time. Further assume this figure shows the TCP with shortest frame time T_1. Since $T_2 \geq T_1$, the deadline of TCP 2, d_2, can occur in the frame T_1 at most once. Now d_2 cannot occur in the interval t_1t_2 since this means that the CPU is idle for some time in that interval; the only exception occurs if $d_2 = T_2$, (since then the CPU can work for TCP_2 in the remainder of t_1t_2); or if d_2 always coincides with either t_1 or t_2, or if d_2 always falls in the interval t_0t_1. To maintain these restrictions it is necessary that any two occurrences of d_2 must be separated by mT_1; therefore $T_2 = mT_1$.

It is tempting to conclude that this condition can be extended to the case of more than two TCPs by requiring $d_i = T_i$ for all i or $T_i = m_iT_1$. A proof is not yet available, however.

Note that the scheduling property just derived is *not* responsible for the failure of IFP to schedule more than about 69 percent of the CPU (Figure 3); the development of that case remains unchanged if in Figure 3 we let all $d_i = T_i$. Thus this failure is peculiar to IFP and, in fact, we will show later that for $d_i = T_i$, the relative urgency (RU) and other algorithms do not suffer from this restriction.

THE MINIMAL TIME SLICING (MTS) ALGORITHM

In implementing ITS, ΔT is assumed to be infinitely small. For any given set of TCPs, a time slicing algorithm which retains most features of ITS with a *finite* ΔT can be developed. This is the minimal time slicing (MTS) policy. It is based on the realization that, for a given set of TCPs, the smallest time unit of interest

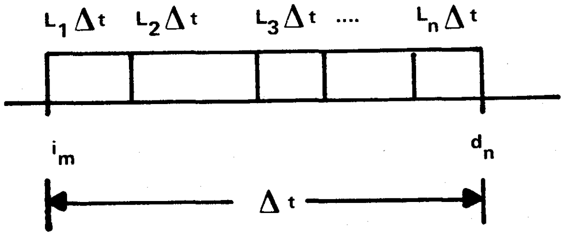


Figure 8—The MTS policy in a scheduling interval. In every scheduling interval Δt, the CPU is allocated in some arbitrary order to all TCPs, such that each receives $L_i \Delta t$ CPU-seconds

from the scheduling viewpoint, called a *scheduling interval*, is the time between the occurrence of two successive "scheduling events." By "scheduling events" we mean events that affect the CPU loading; namely, interrupt instances and deadlines. The scheduler can guarantee that each TCP meets its dealine by allocating the CPU, in *each* scheduling interval, in some arbitrary order, according to the load factors. Figure 8 is a graphic illustration of this policy. The practical implementation of MTS requires all TCPs to be periodic so their interrupt instances are predictable.

MTS suffers from the same weakness as ITS in scheduling processes for which $d_i < T_i$; MTS, however, is a practical algorithm as long as all T_i are known to the scheduler and the context-switching time is small.

Note that both ITS and MTS are *self regulating* in that the CPU time used by each TCP is always under the control of the system so that CPU *protection* is guaranteed to all TCPs. IFP, in contrast, does not have this feature; it requires additional hardware and software safeguards and monitoring. However, the required watch-dog programmable timer and the associated added overhead in context switching are not more complex than those required in implementing the MTS or approximate ITS policies.

THE RELATIVE URGENCY (RU) ALGORITHM

The Relative Urgency (RU) policy, developed by M. S. Fineberg and first reported in Reference 1, is similar to MTS in that it requires the scheduler to reevaluate the scheduling environment on every interrupt. RU differs from MTS in that during the period between two successive interrupts, RU is priority-oriented: the highest priority TCP is given control at the start of that period, and allowed to continue until it completes its computation or until the next interrupt. Priorities are re-allocated on every interrupt and are proportional to deadlines *as they exist at the interrupt instance* (i.e., not simply to their relative lengths); the TCP whose deadline is most imminent (and that has not yet completed its computation for its current frame) receives the highest priority. If a TCP completes its computation during an interval, then the next-highest priority TCP receives the CPU; priority re-evaluation is not required at completion instants.

RU has an obvious advantage over MTS: in general, RU performs less context switching; overhead is therefore smaller under RU.

At least for the case of two TCPs, it can be shown that RU can schedule the CPU at 100 percent efficiency. This is done by showing that, for this case, an RU can always be constructed from an MTS. The proof is carried out by induction.

Assume two TCPs $(T_2 > T_1)$ with $d_i = T_i$, whose interrupts coincide at $t = 0$. Construct an MTS for this case, taking care to, in each scheduling interval, assign the CPU to the TCP whose deadline is nearest. No generality is lost since the order is immaterial under MTS. Let an RU be constructed, and assume that up to scheduling event L, RU and MTS are indeed identical. Consider the event at L and the one following it. There are only three possibilities (see Figure 9): a) at L we have t_1 (interrupt due to TCP 1) and the next event is also t_1, b) $L = t_1$, then t_2; c) $L = t_2$, then t_1. Note that, since $T_2 > T_1$, it is impossible to have two consecutive t_2 without an intervening t_1. Now, for the case (a), under RU we would allocate the CPU at L to TCP 1 (its deadline is nearest); it would execute for $L_1 T_1$ seconds, since the interval $t_1 t_1$ is T_1. Since the CPU is assumed 100 percent scheduled, the remainder of $t_1 t_1$, namely $T(1 - L_1)$ would be allocated to TCP 2. Since $L_1 + L_2 = 1$, $L_2 = 1 - L_1$, so this remainder equals $L_2 T_1$.

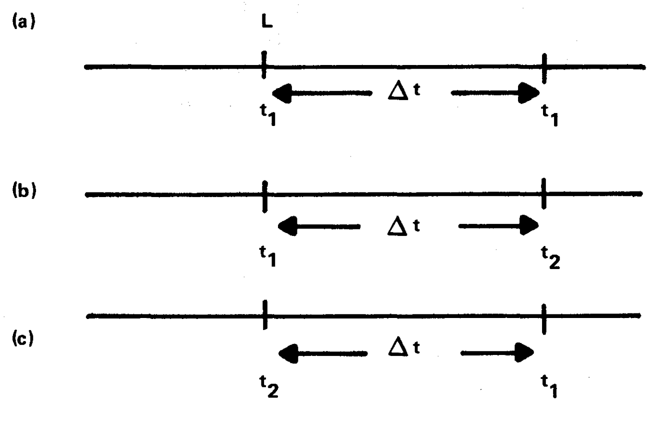


Figure 9—Nomenclature for the proof that MTS and RU are equivalent for two TCPs. The proof is by induction, assuming that up to scheduling event L, RU and MTS are identical, and showing that based on this assumption, RU and MTS must also be identical in the interval Δt

This is precisely the division of t_1t_1, under MTS. For case (b), TCP 2 will receive the CPU, and is short exactly $L_2\Delta t$ CPU-seconds, since by the assumption it received $L_2(T_2-\Delta T)$ CPU-seconds up to event L. This leaves $\Delta t(1-L_2)$ for TCP 1 in Δt, which is equal to $L_1\Delta t$, again showing the equivalence of RU and MTS in Δt. Similar conditions apply in case (c). Finally, at $t=0$ we have case (a) (with $L=t_1=t_2$) but for this case we already showed that RU = MTS. Hence the assertion is proved by induction.

There is considerable empirical evidence that RU is capable of scheduling the CPU at 100 percent efficiency for all T_i/T_{i-1} ratios when all $d_i=T_i$; in this case, RU is at least as efficient as MTS or ITS (more efficient in fact, due to lower switching overhead); when the T_i are not integer multiples of T_1 (the shortest frame time), IFP is known to be less efficient than either MTS, ITS or RU. When $d_i<T_i$, RU is capable of scheduling more of the CPU than MTS or ITS, i.e., more TCPs can be accommodated under RU.

REGARDING MULTIPROCESSING

Considerable effort has been expended on time-critical scheduling techniques for multiprocessor systems (e.g., References 3 and 4). For TCPs of the type considered in this paper, multiprocessor systems do not offer any advantages (if one excludes redundancy or backup requirements). In fact, it appears that in some cases, multiprocessor systems are definitely less desirable than an "equivalent" single processor. By "equivalent" is meant that each of m CPUs execute at a rate $1/m$ as fast as the single CPU.

Given m processors and n TCPs $(m<n)$ with equal C_i and such that all n TCPs can be executed serially on the single, fast processor in time Δt. Then each of the m processor can completely process at most p TCPs, where p is an integer that satisfies $p(\Delta t/n)\ m \overline{<} \Delta t$. In other words, $mp \overline{<} n$, which means that whenever the number of TCPs n is not a multiple of the number of CPUs m, there will remain one TCP that cannot be accommodated within Δt using m processors, whereas the fast processor accommodates all TCPs within Δt. Furthermore, if Grosch's Law holds, the faster CPU should cost less than the m slower units, so that even from the economic standpoint, multiprocessor systems do not appear desirable.

CONCLUSION

This paper presented primarily heuristic extensions and generalizations of concepts that were for the most part first sketched in Reference 1. Much work remains to be done in strengthening and extending the proofs and in discovering the properties of scheduling with I/O constraints. The relation between RU and MTS is particularly intriguing. A consistent theory of scheduling of time-critical processes is within reach.

ACKNOWLEDGMENT

I am indebted to the referees for their excellent, constructive critiques of the original version of this paper.

REFERENCES

1 M S FINEBERG O SERLIN
Multiprogramming for hybrid computation
Proc AFIPS FJCC 1967

2 O SERLIN
CPU scheduling in a time critical environment
Operating Systems Review June 1970 (pub by ACM SIGOPS)

3 R R MUNTZ E G COFFMAN
Preemptive scheduling of real time tasks on multiprocessor systems
J ACM Vol 17 No 2 April 1970 pp 324-338

4 G K MANACHER
Production and stabilization of real time task schedules
J ACM Vol 14 No 3 July 1967 pp 439-465

A class of allocation strategies inducing bounded delays only

by EDSGER W. DIJKSTRA

Technological University
Eindhoven, The Netherlands

We consider a finite set of persons, say numbered from 1 through M, whose never ending life consists of an alternation of eating and thinking, i.e., (in the first instance) they all behave according to the program

```
cycle begin eat;
            think
      end.
```

The persons are living in parallel and their common accommodations are such that not all combinations of simultaneous eaters are permitted. As a result: when a person has finished thinking, some inspection has to take place in order to decide whether he can (and shall) be granted access to the table or not. Similarly, when a person leaves the table, some inspection has to take place in order to discover whether on account of the changed occupancy of the table one or more hungry persons could (and should) be admitted to the table. This situation is reflected by writing their program

```
cycle begin ENTRY;
            eat;
            EXIT;
            think
      end
```

with the understanding that

(1) all inspection processes "ENTRY" and "EXIT" take only a finite period of time and exclude each other in time. (As a result of the postulated mutual exclusion of the inspections "ENTRY" and "EXIT," a "local delay" of person i, wanting to invoke such an inspection, may be needed. We postulate that such a "local delay" will only last a finite period of time—the requests for these inspections could be dealt with on the basis of "first come, first served"—and we shall not mention these local delays any further, because they are now irrelevant for the remainder of our considerations.)

(2) as a result of such an inspection the person invoking it may be put to sleep (i.e., prevented, for the time being, from proceeding with his life as prescribed by the program).

(3) as a result of such an inspection, one or more sleeping persons may be woken up (i.e., induced to proceed with their life as described by the program).

We restrict ourselves to such exclusion rules for simultaneous eaters that

condition 1: if V is a permissible set of simultaneous eaters, so is any subset of V

condition 2: each person occurs in at least one permissible set of simultaneous eaters. (Note: if a person occurs in exactly one permissible set of simultaneous eaters, this set contains—on account of condition 1—only himself and no one else.)

From condition 1 it follows that there is no restriction on the set of simultaneous thinkers; as a result the inspection EXIT will never have the consequence that the person invoking it will be put to sleep. As a result, persons can only be sleeping on account of having invoked the inspection ENTRY and the act of admitting person i to the table can be associated with the waking up of person i. (A little bit more precise: if during the inspection ENTRY as invoked by person i the decision to admit him to the table is *not* taken, he is put to sleep, otherwise he is allowed to proceed. If in any other inspection the decision to admit person i to the table is taken (person i must be sleeping and) person i will be woken up).

Furthermore we restrict ourselves to the case that

condition 3: for each person the action "eat" will take a finite period of time (larger than some positive lower bound and smaller than some finite upper bound); this in contrast to the action "think" that may take an infinite period of time.

In the simplest strategy no inspection will leave a person sleeping whose admittance to the table is allowed as far as the occupancy of the table is concerned. Alas, such a strategy may have the so-called "danger of individual starvation," i.e., although all individual eating actions take only a finite period of time, a person may be kept hungry for the rest of his days. (The classical configuration showing this phenomenon is the Problem of the Dinig Quintuple. Here five places are arranged cyclically around a round table and each of five persons has his own place at the table. The restriction is that no two neighbors may be eating simultaneously. The rule will then be that every person is admitted to the table as soon as he is hungry and none of his two neighbors is eating. In this particular example this rule leaves no choice, i.e., when I leave the table the decisions whether my lefthand neighbor and my righthand neighbor have to be admitted to the table are independent of each other. In this example my two neighbors can starve me to death, viz., when my eating lefthand neighbor never leaves the table before my righthand neighbor is eating and vice versa. If the remaining two persons remain thinking, access to the table will never be denied to my neighbors and with me hungry the process can continue forever.) The moral of the story is that if we are looking for strategies without the danger of individual starvation, we must in general be willing to consider allocation strategies in which hungry persons will be denied access to the table in spite of the circumstance that the occupancy of the table is such that they could be admitted to the table without causing violation of the given simultaneity restrictions. Our interest in strategies without the danger of individual starvation was aroused by the sobering experience that quite a few intuitive efforts to exorcize this danger led to algorithms that turned out to be quite ineffective because they could lead to deadlock situations. The remaining part of this paper deals with a general characterization of strategies that do not contain the danger of individual starvation. We shall restrict ourselves to strategies where the decision to admit persons to the table will be taken for one person at a time; from our analysis it will follow then that our characterization can be given independent of the specific simultaneity restrictions (provided of course they satisfy our stated conditions 1 and 2).

We start by proving a theorem, in which we consider the following (possible) properties of a strategy.

property A: the existence of at least one sleeping person implies at least one person who is eating or leaving the table

property B: for any person i it can be guaranteed that during a period of his hungriness the decision to admit someone else to the table will not be taken more than N_i times, where N_i is a given, finite upper bound for person i.

Our theorem asserts that when conditions 1, 2 and 3 are fulfilled, properties A and B are the necessary and sufficient conditions for any strategy in order not to contain the danger of individual starvation.

The necessity of property A follows from the inadmissibility of the situation in which one or more persons are sleeping while all remaining ones (if any) are thinking. The thinking ones may go on thinking forever, as a result no new inspections will be evoked and the sleeping ones remain hungry for an infinite period of time.

We say that the danger of individual starvation is absent when the hungriness of any person will never last longer than a given, finite period of time. The minimum time taken by the act of eating imposes an upper bound on the personal frequency with which any given person can be admitted to the table; the total number of persons is M and therefore there is an upper bound on the total frequency with which someone is admitted to the table. Therefore the number of admissions during a period of hungriness of person i must always be less than a fixed, finite value: property B is necessary in the sense that a set of fixed, finite N_i's exists such that it is satisfied.

Next we show that the conditions are sufficient. When person i becomes hungry—by invoking ENTRY—we have to show that his hungriness will only last a finite period of time. If in the course of that very inspection he is admitted to the table, it is true (for inspections take only a finite period of time), otherwise he goes to sleep. At the end of that inspection at least one person is eating (on account of property A). From the fact that the action "eat" takes only a finite period of time, the persons now eating will have finished doing so and will have left the table within a finite period of time. From this and property A it follows that within a finite period of time a new person will have been admitted to the table. The assumption that person i remains hungry forever implies that the lucky person must have been someone else, i.e., within a finite period of time the number of times it has been decided during hungriness of person i that someone else is admitted to the table is increased by one. Then the argument can be repeated and within a finite period of time the number of times someone else is admitted to the table would exceed N_i, contrary to our property B. Therefore person i will not remain hungry forever.

Having established that properties A and B are necessary and sufficient for the absence of the danger of individual starvation, we are now in a position to characterize all strategies satisfying them with *a priori* given bounds N_i. For this purpose we associate with each hungry person a counter called "*ac*" (short for "allowance count"). Whenever person i becomes hungry, his *ac* is added to the set of *ac*'s with the initial value$=N_i$; whenever it is decided that a person is admitted to the table, his *ac* is taken away from the set of *ac*'s and all remaining *ac*'s are decreased by one. Property B is guaranteed to hold when no *ac* becomes negative.

We call the set of *ac*'s "safe" when for all $k \geq 0$ holds that at most k *ac*'s have a value$<k$. Note that also the empty set is safe. (Another formulation of safety is that it must be possible to order the *ac*'s, if present, in such a fashion that the first $ac \geq 0$, the second $ac \geq 1$, the third $ac \geq 2$, etc.) Such a safe set has four important properties.

Property 1: No safe set contains a negative element (substitute $k=0$ in the first definition).

Property 2: If removal of an element from the set is accompanied by a decrease of the remaining elements by one, each non-empty safe set contains at least one element that can be removed such that the remaining set is again safe: for this purpose it is sufficient—although one has often greater freedom—to choose one of the smallest values.

Property 3: In the case of an unsafe set, let K be the minimum value of k, such that more than k elements have a value less than k. Addition of a new element to an unsafe set will never make it into a safe one, nor will it lead to an increase of K.

Property 4: In the case of an unsafe set, K (as defined in the previous paragraph) is an upper bound for the number of times that an element can be removed (again each removal being accompanied by a decrease of the remaining elements by 1) before negative values occur.

Properties 3 and 4 tell us that an unsafe set of *ac*'s is bound to lead to negative *ac*'s before it is empty. Therefore safety of the set of *ac*'s must be maintained if we wish to guarantee property B. This imposes a lower bound on the initial values of the *ac*'s, i.e., the N_i. With M processes, at most $M-1$ will be sleeping (property A), if we impose

$$N_i \geq M-2$$

we can guarantee that, given a safe set of *ac*'s, the addition of a new *ac* will never lead to unsafety.

We can now characterize all strategies satisfying properties A and B. We call a person "admissible" if the following three conditions all hold

(1) he must be hungry

(2) his addition to the set V of eating persons would not cause violation of the simultaneity restrictions

(3) his removal from the set of hungry persons would leave a safe set of *ac*'s. We now characterize *all* strategies enjoying our (necessary and sufficient) properties A and B in terms of a *general permission* and a *specific obligation.*

Each inspection ENTRY or EXIT has the general permission to decide (zero or more times) to admit an admissible person to the table. However, those inspections that would violate property A in the case of zero admissions have the specific obligation to admit at least one person to the table. They are the ENTRY while there are no eating persons and the EXIT in which a person is the last to leave the table while there are sleeping persons. In both cases conditions 1 and 2 and property 2 of safe sets guarantee the existence of at least one admissible person.

It is clear that any such strategy will satisfy properties A and B, it is also clear that any other strategy will have to be rejected: if the general permission is violated, an erroneous admission to the table takes place or we end up with either deadlock or violation of property B, if the specific obligation is not fulfilled property A is violated. In this sense we have characterized *all* strategies satisfying properties A and B.

CONCLUDING REMARKS

For a very specific reason the result obtained seems significant. When one is making an operating system one is faced with "absolute requirements" (of a rather logical nature) on the one hand and "desires" (not necessarily all compatible with each other) on the other. In the early design phase of the Multiprogramming System we had the hope that all allocation strategies could be factored in the sense that first we could produce the code that would ensure non-violation of the absolute requirements, in which then all sorts of strategic routines could be plugged in, the idea being that a change of strategic routines could influence the desirability of the systems behavior but could never lead to violation of the absolute requirements. In the later design stages we have not been able to reach that goal: we turned up with allocation strategies for which we could prove that the absolute requirements would never be violated, but each new proposed strategy required a new proof of this fact. The origin of this failure was our inability at that time to give a constructive characterization of *all* possible strategies that were guaranteed to meet our absolute requirements. This paper shows that—at least in the case of the chosen absolute requirements—such a characterization can be given in terms of permissions and obligations and in such a way that it has been proved that the obligation can always be fulfilled. The question under which other circumstances such characterizations can be given is now open for investigation.

ACKNOWLEDGMENTS

The author wishes to express his great indebtedness to A. Martin and W. H. J. Feijen, for it is due to their persistence, in a common effort to clear up our thinking about starvation, that I was guided towards the approach described in this paper.

On modeling program behavior

by PETER J. DENNING

Princeton University
Princeton, New Jersey

INTRODUCTION

This is a paper about the history of the working set model for program behavior. It traces briefly the origins and bases of the idea and some of the results subsequently obtained. The physical context is a hierarchical memory system consisting of a severely limited quantity of main (directly-addressable) storage and an essentially unlimited quantity of secondary (backup) storage. In this context, the intuitive notion of "working information" as the set of words which are (or should be) loaded in main memory at any given time in order that a program may operate efficiently is as old as programming itself.[1] The sharply increased interest in program models since the mid-1960s is a direct consequence of the widening use of virtual memory and multiprogramming techniques, which have shifted the responsibility of memory management from programmers to machines. I am assuming here that the purpose of memory management is ensuring that an active program's working information is present in main memory, and the purpose of a program model is providing a basis for determining a program's working information at a given time and predicting what it will be at a future time.

The intuitive notion calls for "working information" to be defined as machine-independent as possible, so that there can be an "absolute" measure of a program's memory demand. Moreover, the definition should be dynamic since we wish ultimately to derive adaptive memory management policies from it. In my own efforts to formalize the intuitive notion, I adopted the spirit of an approach suggested as early as 1965, according to which a program's "working set" at a given time consists of those objects which must be loaded in main memory in order to guarantee a given level of processing efficiency.[19] It seemed to me that an ideal definition should, either explicitly or implicitly, specify a priority listing on a program's information objects at each moment of time; then, given the desired level of processing efficiency and the physical characteristics of the system (e.g., the ratios of speeds between various levels of memory, or the capacities of various levels of memory), one should be able to compute, at any given time, an integer k such that the k objects of highest priority constitute the program's working set at that time. Because a direct formalization of this idea did not appear immediately amenable to analysis, I settled instead on a definition of "working set" as a collection of recently referenced objects.[11,12] As will be seen, this alternative definition is equivalent to the former one under realistic assumptions about programs.

A useful definition of working set should, in addition to the above, satisfy two properties. First, it should be based on easily observable properties of program behavior. As suggested in Figure 1, the stream of addresses (the *address trace*) generated by a processor is perhaps the most easily observed aspect of dynamic program behavior. Now, a given program's instruction and data code is ordinarily divided into blocks of various sizes, each block being identified by a number and consisting of words with contiguous addresses. Since memory allocation policies deal with blocks, not words, as the units of allocation, there is no loss of generality in supposing that a string of block numbers (the *reference string*) represents observable program behavior. Specifically, if $a_1a_2 \ldots a_t \ldots$ is an address trace in which a_t is the address referenced at time t in the time frame of the program (program time is measured discretely, $t=1, 2, 3, \ldots$), and r_t is the number of the block containing address a_t, the reference string is $r_1r_2 \ldots r_t \ldots$. In the discussion to follow, I shall assume that the blocks of a program are all of the same size (i.e., paging is used); this is purely a matter of convenience, as generalizations to variable block size are straightforward. Under this assumption, a reference string is a sequence of page numbers.

Second, the definition of working set should be based

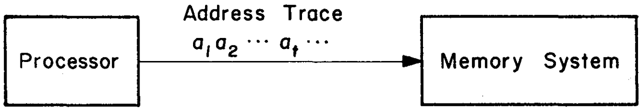


Figure 1—Address trace

on information about a program's observed past and not on information about its future—i.e., the working set at time t should depend only on $r_1 r_2 \ldots r_t$ and not on $r_{t+1} r_{t+2} \ldots$. I imposed this requirement because I wanted a program model which would guide a choice of memory management policy in the absence of prior knowledge of program behavior (e.g., programmer or compiler advice or predictions[1]), since experience suggests that prior knowledge is either inordinately expensive to obtain or is unreliable. I was not attempting to deny that, if reliable prior information is present, it can and should be used to improve an estimate of a program's working set; I wanted only to avoid making prior information a fundamental assumption of the model.

A DEFINITION OF WORKING SET

With these general notions about how "working set" should be defined, one can proceed to develop the definition itself. As will be seen, the viability of the "most recently referenced pages" definition depends on the property of *locality* (known also as "locality of reference") which is exhibited to varying degrees by all practical programs.[2,3,5–8,12,13,15,17,24,26] The property of locality can be summarized as three statements:

L1. A program distributes its references nonuniformly over its pages, some pages being favored over others.
L2. The density of references to a given page tends to change slowly in time.
L3. Two reference string segments are highly correlated when the interval between them is small, and tend to become uncorrelated as the interval between them becomes large.

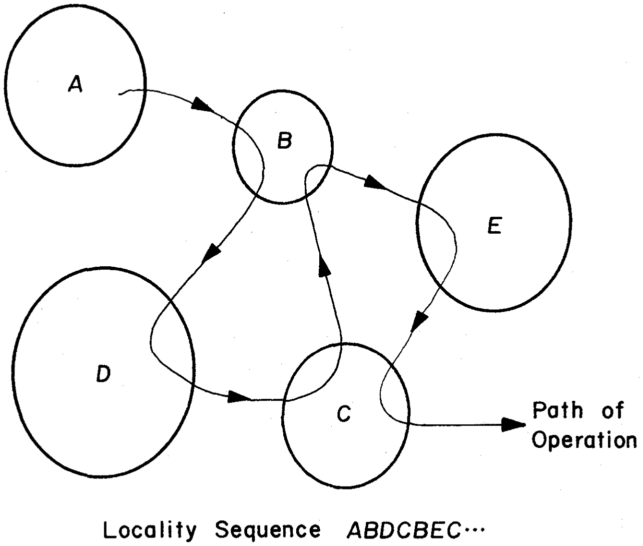
These properties are abstractions of commonly observed phenomena: programs tend to reference pages unequally, they tend to cluster references to certain pages in short time intervals, they can be run efficiently in memory spaces considerably smaller than the program size.[2,21,26] Locality derives from the behavior and style of programmers: they tend to use sequential and looping control structures frequently, they tend to group data into content-related blocks, and they tend to concentrate on small parts of large problems for moderately long intervals.[5–7,15,24,26] The degree to which locality is exhibited is strongly a function of programming style—i.e., algorithm strategy and data organization;[5–7,26] it is reported that high degrees of locality can often be achieved at a cost of under 5 percent of total programming costs.[26] As will be seen below, the definition of working set as a collection of recently referenced pages improves as an estimator of a program's working information as the degree of locality increases.

Based on the foregoing ideas, one can formulate a model for locality. Suppose P is the set of pages of a given program. Associated with the given program is a collection of *localities*, each a distinct subset of P; we denote them by $A, B, C, \ldots$. Two localities may overlap. As suggested in Figure 2, one may imagine that the executing program traces a path of operation through its localities; the path defines a sequence of localities, e.g., in Fig. 2 it is

$$L_1 L_2 L_3 \ldots L_i \ldots = A\ B\ D\ C\ B\ E\ C \ldots$$

Now, let t_i denote the length of time (the *residence time*) the program spends in the ith locality L_i of the locality sequence; while the path is in locality L_i, the program generates a reference substring of length t_i, consisting of pages from L_i only. As an example, consider a program with pages $P = \{1, 2, \ldots, 6\}$ and three localities

$$A = \{1, 2, 3\} \qquad B = \{4, 5, 6\} \qquad C = \{1, 2, 6\}$$



Locality Sequence *ABDCBEC*···

Figure 2—Locality

Suppose the locality sequence is $L_1L_2L_3=ABC$ and the residence times are $t_1t_2t_3=656$. A reference string consistent with these conditions is

$$\underbrace{1\,3\,2\,1\,2\,3}_{A}\quad\underbrace{6\,4\,5\,6\,4}_{B}\quad\underbrace{6\,1\,2\,6\,2\,1}_{C} \qquad (1)$$

Based on the property of locality, one would expect that the residence times tend to be long and that neighboring localities (on the path of operation) tend to overlap.

Ideally, a program's working set at any given time should comprise exactly the pages from the current locality at that time. In practice (at least without the services of a Delphic Oracle) the working set will be an estimate of locality based on observations of the recent past reference string. Accordingly, if $r_1r_2\ldots r_t\ldots$ is a reference string, we define the *working set* $W(t, T)$ at time t to be the set of distinct pages referenced among the T most recently referenced pages, i.e., among $r_{t-T+1}\ldots r_t$. If $t<T$, $W(t, T)$ consists of the distinct pages among $r_1r_2\ldots r_t$ and if $t<0$, $W(t, T)$ is empty. The parameter T is called the *window size,* since $W(t, T)$ can be regarded as the contents of a window looking backward at the reference string. The working sets for $T=4$ on the reference string (1) given above are shown in Figure 3.

The working set is *correct* when it comprises exactly the pages of the current locality. The correctness depends on the choice of window size T; in general, two conditions would have to be satisfied:

1. T is sufficiently large that every page of a locality is in the working set with high probability.
2. T is comparable to or much less than the average locality residence time, i.e., the probability of the window's containing more than one interlocality transition is small.

Locality	Page	Refs.	1	3	2	1	2	3	6	4	5	6	4	6	1	2	6	2	1
AC	1		x	x	x	x	x	x	x						x	x	x	x	x
AC	2				x	x	x	x	x	x						x	x	x	x
A	3			x	x	x	x	x	x	x	x								
B	4									x	x	x	x	x	x	x			
B	5										x	x	x	x					
BC	6								x	x	x	x	x	x	x	x	x	x	x
Correctness -					x	x	x	x				x	x				x	x	x
Size -			1	2	3	3	3	3	4	4	4	3	3	3	3	4	3	3	3

Figure 3—Working sets for $T = 4$

The higher the degree of locality exhibited by the program, the more likely it is that T can be chosen to satisfy the two conditions, and consequently the larger the fraction of time during which the working set is a correct estimate of current locality. Additional factors influencing the choice of T will be mentioned shortly.

The use of a moving window to estimate locality as discussed above is more a matter of convenience than necessity. In a practical implementation it would be sufficient to associate "use-bits" with pages; a page's use-bit is set when the page is referenced, and is read and reset at the end of each interval of T references. The use-bits found to be set at the end of a T-interval would define a working set, and the working set so measured would be used as an estimator of locality until updated at the end of the subsequent T-interval. Indeed, my original proposal for working sets was based on this idea.[10] The T-intervals can coincide with the time slices used to schedule programs.[9,11,12,22,23,27]

Suppose one postulates a "working set paging algorithm"—i.e., a page replacement algorithm that replaces only the non-working set pages of a program. It can be demonstrated that this algorithm will implement the "principle of optimality" for page replacement (replace the page not expected to be used again for the longest time) provided locality is good.[15] In other words, the use of locality as a basis for memory management is close to being optimal.

THE WORKING SET PRINCIPLE AND THRASHING

The *working set principle* for memory management states that a program may be active (i.e., eligible to receive time slices on a processor) only if its working set is loaded in main memory.[11,12,17] It follows from this statement that the number of active programs is limited by the main memory capacity and the sizes of working sets; indeed, if there are n active programs, the main memory capacity is M pages, and the ith working set $W_i(t, T_i)$ contains w_i pages, the working set principle states that the relation

$$w_1+\cdots+w_n\leq M \qquad (2)$$

must be true with high probability at all times. It follows from this statement that only the pages not in the working set of any active program are subject to removal from main memory; in particular, if every page in memory were a member of a working set, it would be necessary to deactivate some program in order to create candidate pages for removal.

If the working set principle is not followed, it is possible for attempted overcommitment of memory to precipitate *thrashing,* a collapse of system performance, especially when the ratio of speeds between auxiliary and main memory is high.[9,12,13,15,23,27] It is of course possible for the working set principle to be violated without having thrashing; the point is that the principle defines a sufficient condition for avoiding thrashing.[13,15]

An efficient implementation of the principle would employ a compromise between the ideas of demand paging and swapping. Demand paging would be used to acquire pages during a program's active intervals, and swapping would be used to move a working set as a unit at the beginnings and ends of active intervals. Such strategy can be remarkably effective when properly designed.[20]

WORKING SET SIZE AND PAGING RATE

The working set size $w(t, T)$ is the number of pages in the working set $W(t, T)$. The average size for the first k references is

$$s_k(T) = \frac{1}{k} \sum_{t=1}^{k} w(t, T) \tag{3}$$

and the limiting *average size* is

$$s(T) = \lim_{k\to\infty} s_k(T) \tag{4}$$

(The error introduced by using $s(T)$ when in practice one would use $s_k(T)$ for some k is not significant under the assumptions of locality.[17]) Consider a reference string segment $r_1r_2 \ldots r_k$; for this segment, let $q(k)$ denote the number of times r_{t+1} was not in $W(t, T)$, $0 \leq t < k$. The *missing-page rate* is

$$m(T) = \lim_{k\to\infty} \frac{q(k)}{k} \tag{5}$$

so that $m(T)$ measures the average rate at which pages are entering the working set. Note that $m(T)$ is an upper bound on the page-fault rate of a single program in a system using the working set principle of memory management, for a page may leave the working set and subsequently return without leaving main memory in the meantime.

The important properties of $s(T)$ are summarized in Figure 4. The curve $s(T)$ is upper-bounded by the smaller of T and N, where N is the size of the program; $s(T)$ is nondecreasing in T, i.e., $s(T) \leq s(T+1)$; $s(T)$ is concave downward; and the "slope" of $s(T)$ is the missing-page rate:

$$s(T+1) - s(T) = m(T) \tag{6}$$

Finally, $m(T)$ is nonincreasing, i.e., $m(T) \geq m(T+1)$. The proof of (6) is straightforward: note that

$$w(t+1, T+1) = w(t, T) + \Delta w \tag{7}$$

where Δw is 1 if the next reference r_{t+1} is not in $W(t, T)$ and 0 otherwise, i.e., Δw is 1 if and only if the next page referenced is missing. Observe that the expected value of Δw is $\overline{\Delta w} = m(T)$. Taking expectations on both sides of (7), one finds

$$s(T+1) = s(T) + \overline{\Delta w} = s(T) + m(T) \tag{8}$$

which is equivalent to (6). A complete discussion of these properties, together with proofs, can be found in Reference 17. Experimental verification is presented in Reference 25. Further relations between $s(T)$ and $m(T)$ and page "interreference-interval distributions" can be found in Reference 17.

PRACTICAL IMPLEMENTATIONS OF WORKING SET POLICIES

There is a close kinship between the working set model and LRU (least-recently-used) paging, a kinship which can be used to approximate the working set model in systems where implementing a window T is impractical. The LRU paging algorithm is a demand-paging algorithm for managing a main memory space held fixed at k pages; at each page fault the least-recently-used page is replaced to make way for the page entering memory, so that the memory always contains the k most recently used pages. By comparison, a working set

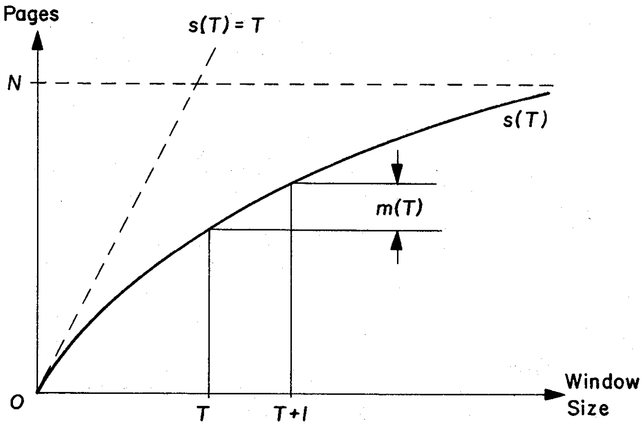


Figure 4—Mean working set size

$W(t, T)$ always contains $w(t, T)$ most recently used pages, where $w(t, T)$ is variable. It follows immediately that the $s(T)$ and $m(T)$ curves of a program can be used to estimate the page-fault rate $F(k)$ of the LRU algorithm in a k-page memory; see Figure. 5. A more detailed argument[17] shows that, under general conditions, $F(k) \geq m(T_k)$.

Now, one can imagine implementing a working set policy according to which the window size T is varied dynamically so that the missing-page rate $m(T)$ is controlled to be within a range m_1 to m_2, where $m_1 > m_2$. The importance of such a policy is twofold: not only does it imply each program operates in a known range of processing efficiency, but it allows a system designer to determine bounds on the paging traffic generated by a given statistical mix of active programs. The policy determines a sequence of times $t_0 t_1 t_2 \dots t_i \dots$ such that $T_i = t_i - t_{i-1}$ is the ith value of window size and (t_{i-1}, t_i) is the ith "sampling interval." During the ith sampling interval the program is run under a working set policy with window size T_i, the number p_i of pages entering the working set during that interval is counted, and p_i/T_i serves as an estimate for the missing page rate $m(T_i)$ during that interval. If p_i/T_i exceeds m_1 then T_i was too small and T_{i+1} is set to a value larger than T_i; if p_i/T_i is in the range m_1 to m_2, T_{i+1} is the same as T_i; and if p_i/T_i is smaller than m_2 then T_i is too large and T_{i+1} is set to a value smaller than T_i.

By virtue of the relation between the working set model and LRU paging, one can approximate the foregoing procedure using LRU paging in a memory space whose size is varied from time to time. In this case, T is taken as a fixed sampling interval (e.g., a time slice), p the number of page faults in the previous sampling interval, and p/T the estimate of page fault rate. If p/T exceeds m_1, the program is allocated one additional page of memory during the next sampling interval (this page being filled on demand at the next fault); if p/T is in the range m_1 to m_2, the memory allocation is unchanged; and if p/T is smaller than m_2, the memory allocation is reduced one page during the next sampling interval (the least-recently-used page being removed). This process is repeated for each sampling interval. During each interval the memory space allocation is fixed and LRU paging is employed. At any time the contents of memory serve as an estimate of the working set.

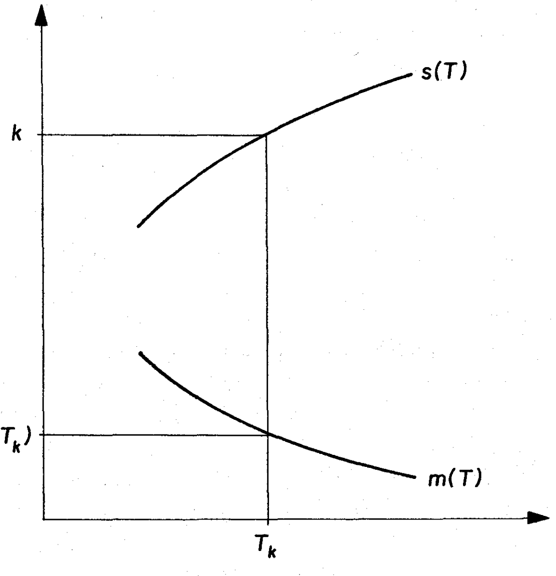


Figure 5—Estimating LRU paging rate

It is important to note that, if the LRU approximation to the working set principle is used in a multiprogrammed memory, a page fault generated by a program causes a page to be replaced from that program's own working set but never from another program's working set. Put another way, the LRU algorithm is applied on a "local" basis but not on a "global" one. Thus a program's missing-page rate depends only on its own behavior and never on that of another program: its working set survives intact despite the paging activities of other programs with which it shares main memory.

DISTRIBUTION OF WORKING SET SIZE

It can be shown[17] that whenever the third property L3 of locality holds—reference substrings tend to become uncorrelated as the interval between them becomes large—the working set size is approximately normally distributed about its mean $s(T)$. Suppose $w(t, T)$ is a working set size process with mean s and variance σ^2. The normal approximation for the distribution of $w(t, T)$ asserts that, for large t,

$$\Pr[w(t, T) \leq x] = \Phi\left(\frac{x-s}{\sigma}\right) \tag{9}$$

where

$$\Phi(x) = \frac{1}{(2\pi)^{1/2}} \int_{-\infty}^{x} e^{-u^2/2}\, du \tag{10}$$

is the cumulative distribution of a normally distributed random variable with zero mean and unit variance. The ability to consider $w(t, T)$ as a normally distributed random process is of key importance, in view of the enormous literature and theory extant about such processes.[8]

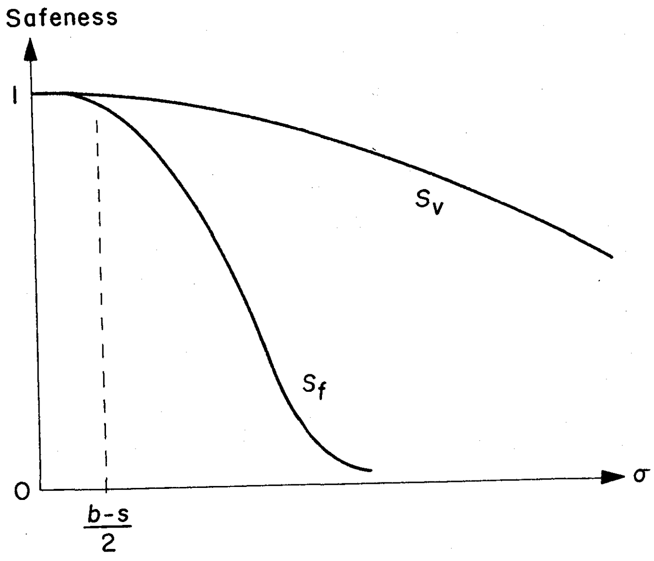


Figure 6—Safeness of fixed and variable partitioning

Two deviations are inherent in the use of the normal approximation to the distribution of working set size: the normal distribution is continuous whereas working set size is discrete, and the normal distribution is defined for negative values. The first deviation is not significant for programs whose working set size varies over a wide enough range of values, and the second is not significant for most parameter values (s and σ) of interest.

The normal approximation has been found remarkably accurate when the correlation property (L3) of locality holds.[8] If this property fails to hold (e.g., the program contains a global loop), it will be possible to predict arbitrarily distant future reference patterns on the basis of present ones and significant deviations from the normal approximation will be observed.[25]

FIXED VERSUS VARIABLE PARTITIONING OF MEMORY

The strategies for allocating main memory, under multiprogramming, usually lie near one of two extremes. According to the *fixed partition* strategy, each of the programs sharing the memory is allocated a fixed amount of memory for its private use. According to the *variable partition* strategy, the amount of space allocated to each program depends on its working set size. The principal argument advanced for fixed partitioning is simplicity of implementation; a recent study by Coffman and Ryan, however, suggests that the cost of implementing variable partitioning is more than adequately compensated by the saving in memory requirement.[8]

Under the assumption that n programs whose memory demands are normally distributed working set processes with the same mean s and variance σ^2, Coffman and Ryan compared a fixed partition and a variable partition strategy with respect to a "safeness factor" as a performance measure (the safeness factor is related to the volume of paging and overall efficiency). In both cases the total mount of main memory was M pages. In the fixed partition case, each program is allocated a block of b pages and the total memory capacity is $M=nb$; a block is considered safe (unsaturated) whenever the instantaneous value of working set size for the program using that block does not exceed b, and the multiprogrammed memory is considered safe whenever all the blocks are safe at the same time. In the variable partition case, each program is allocated an amount of space equal to its working set size; this system is considered safe if the sum of the working set sizes does not exceed the memory capacity M. The values of S_f, the safeness factor for fixed partitioning, and S_v, the safeness factor for variable partitioning, are readily determined under the assumption of normally distributed working set size. The results are suggested in Figure 6. Whenever the variance σ^2 of working set size is small enough so that the block size b is at least $s+2\sigma$ [equivalently, σ is less than $(b-s)/2$], the two strategies are competitive; but when σ becomes large, the variable partition strategy is considerably safer than the fixed partition strategy.

Under the assumption of variable partitioning and a working set memory management policy, it is possible to specify the equipment configuration (relative amounts of processor and memory resources) of a computer system.[14]

CONCLUSIONS

Originally introduced as a means of understanding the vagaries of multiprogrammed memory management, the working set model has proved useful both as an aid for guiding one's intuition in understanding program behavior, and as a basis for analyses and simulations of memory problems. To summarize: it has led to a sharpening of the notions "working information" and "locality of reference"; it has aided in the development of multiprogrammed memory management strategies which guarantee each program a given level of efficiency and which prevent thrashing; it has quantified the relationship between memory demand and paging rate; it has led to better understanding of paging algorithm behavior in both demand and prepaging modes; and it has enabled preliminary analysis of fixed *v.* variable partitioning of multiprogrammed memories and of system equipment configurations. Approximations to the working set principle have been used successfully in many contemporary systems both large and small, demonstrating the viability of the model: systems reporting success include the RCA Spectra 70/46,[9,23,27] certain TSS/360 installations,[18] the TENEX system for the PDP-10,[4] and MULTICS. Many systems have used, successfully, approximations to the working set model, and at least one is including hardware capable of implementing the moving-window definition of working set.[22]

Many extensions to the model are possible and worthy of future investigation. For example, the working set can be defined as the set of most recently used blocks of program code, the assumption of fixed sizes pages being eliminated. Or, it may be useful in some situations to partition the working set into two disjoint subsets, the instruction working set and the data working set, and to implement the working set principle separately for both these working subsets. Or, none of the foregoing considerations has allowed for the possibility of sharing—overlapping working sets—in certain cases the effects of sharing have been quantified, a notable reduction in total memory demand being observed.[12]

REFERENCES

1 Proceedings of a Symposium on Storage Allocation
ACM Comm ACM 4 10 October 1961

2 L A BELADY
A study of replacement algorithms for virtual storage computers
IBM Sys J 5 2 1966 pp 78-101

3 L A BELADY C J KUEHNER
Dynamic space sharing in computer systems
Comm ACM 12 5 May 1969 pp 282-288

4 D G BOBROW J D BURCHFIEL D L MURPHY R S TOMLINSON
TENEX—A paged time sharing system for the PDP-10
Comm ACM 15 3 March 1972

5 B BRAWN F GUSTAVSON
Program behavior in a paging environment
AFIPS Conference Proceedings Vol 33 Fall Joint Computer Conference 1968 pp 1019-1032

6 B BRAWN F GUSTAVSON
Sorting in a paging environment
Comm ACM 13 8 August 1970 pp 483-494

7 E G COFFMAN JR A C McKELLAR
Organizing matrices and matrix operations for paged memory systems
Comm ACM 12 3 March 1969 p 153ff

8 E G COFFMAN JR T A RYAN
A study of storage partitioning using a mathematical model of locality
Comm ACM 15 3 March 1972

9 W M DEMEIS N WEIZER
Measurement and analysis of a demand paging time sharing system
Proc 24th National Conference ACM August 1969
pp 201-216

10 P J DENNING
Memory allocation in multiprogrammed computer systems
MIT Project MAC Computation Structures Group Memo No 24 March 1966

11 P J DENNING
The working set model for program behavior
Comm ACM 11 5 May 1968 pp 323-333

12 P J DENNING
Resource allocation in multiprocess computer systems
MIT Project MAC Technical Report MAC-TR-50 PhD Thesis May 1968

13 P J DENNING
Thrashing—Its causes and prevention
AFIPS Conference Proceedings Vol 33 Fall Joint Computer Conference 1968 pp 915-922

14 P J DENNING
Equipment configuration in balanced computer systems
IEEE Trans Comp C-18 11 November 1969 pp 1008-1012

15 P J DENNING
Virtual memory
Computing Surveys 2 3 September 1970 pp 153-189

16 P J DENNING
Third generation computer systems
Computing Surveys 3 4 December 1971

17 P J DENNING S C SCHWARTZ
Properties of the working set model
Comm ACM 15 3 March 1972

18 W DOHERTY
Scheduling TSS/360 for responsiveness
AFIPS Conference Proceedings Vol 37 Fall Joint Computer Conference 1970 pp 97-112

19 E L GLASER J F COULEUR G A OLIVER
System design for a computer for time sharing application
AFIPS Conference Proceedings Vol 27 Fall Joint Computer Conference 1965 pp 197-202

20 M L GREENBERG
An algorithm for drum storage management in time sharing systems
Proceedings 3rd ACM Symposium on Operating Systems Principles October 1971

21 J S LIPTAY
The cache
IBM Sys J 7 1 1968 pp 15-21

22 J B MORRIS
Demand paging through utilization of working sets on the Maniac II
Los Almos Scientific Labs Los Alamos New Mexico 1971

23 G OPPENHEIMER N WEIZER
Resource management for a medium scale time sharing operating system
Comm ACM 11 5 May 1968 pp 313-322

24 B RANDELL C J KUEHNER
Demand paging in perspective
AFIPS Conference Proceedings Vol 33 Fall Joint Computer Conference 1968 pp 1011-1018

25 J RODRIGUEZ-ROSELL
Experimental data on how program behavior affects the choice of schedular parameters
Proceedings 3rd ACM Symposium on Operating Systems Principles October 1971

26 D SAYRE
Is automatic folding of programs efficient enough to displace manual?
Comm ACM 13 12 December 1969 pp 656-660

27 N WEIZER G OPPENHEIMER
Virtual memory management in a paging environment
AFIPS Conference Proceedings Vol 34 Spring Joint Computer Conference 1969 pp 234ff

Magnetic disks for bulk storage—Past and future

by JOHN M. HARKER and HSU CHANG

IBM Systems Development Division
San Jose, California

INTRODUCTION

In the early days of electronic data processing, new application requirements arose that could not be met adequately by batch processing, a mode dictated by the sequential nature of card and tape input-output equipment. Users needed a device to store and directly access relatively large amounts of data on-line. The first such products, IBM's RAMAC disk file and Sperry Rand's Randex drum, partially answered these needs. However, their use could be justified only for storage of highly active data due to their limited capacity and high cost compared to off-line tape storage.

The subsequent introduction of the interchangeable disk pack lowered disk storage costs and offered a means of supplementing on-line disk capacity with shelf storage of infrequently used data. Systems designed with this component, in combination with programming support for skip sequential processing and other new features, made disk-oriented data processing commonplace by the middle of the 1960s. During the past five years, the acceptance of the disk pack has continued to be a major factor in the explosive growth of disk storage. While important developments have also taken place in fixed media disk devices, the total on-line capacity of disk pack devices today is more than several hundred times that of fixed media files and drums.

A compounding of four other factors had a substantial influence on the success of the disk pack and the rapid growth of disk storage during this period. First, the total data processing base expanded at a very high rate following delivery of the IBM System/360 in 1965. Second, the introduction of IBM's multidrive 2314 disk facility made hundreds of megabytes of on-line storage practical at a cost per byte significantly below that of previous disk pack devices. Third, large programming operating systems became widespread, permitting a user to conveniently incorporate disk processing in his system configuration without knowledge of the device characteristics or programming requirements. Last, remotely operated time-sharing networks and the evolving capability of both hardware and systems programs to handle multiple jobs efficiently put a high premium on having direct access to a large data bank. Not to be overlooked, the operating systems themselves required an increasing amount of on-line storage capacity, typically 25 megabytes at a large installation.

In this systems-oriented environment, the disk pack still performs a vital function in system reliability by assuring that data availability is not jeopardized by the failure of a single drive. However, the use of disk packs as a way of loading and unloading user jobs can be expected to diminish at installations with extensive operating systems since these systems view the peripheral hardware as a resource which is allocated and scheduled without operator intervention. For efficient system usage, a single user's data sets are frequently spread among several packs (or "volumes") of storage and may be interspersed with data associated with other user jobs.

The gradual shift in data management from user to system began with the increase in disk pack capacity from approximately 7 megabytes in the IBM 2311 disk drive to 30 megabytes in an IBM 2314 drive. This trend will accelerate as users move to new disk devices with considerably larger capacity such as the 100-megabyte disk pack in the IBM 3330 drive (Figure 1).

Still another important development over the past few years is an increase in the number of disk device manufacturers whose resources and competition can stimulate further advances.

To see where these trends will lead, it is necessary to understand what key technology developments can be expected, and then put them in the context of future system requirements. Since present disk storage devices rest on a technology base of magnetic recording, primary emphasis will be placed on this technology and its potential for further progress. Alternative tech-

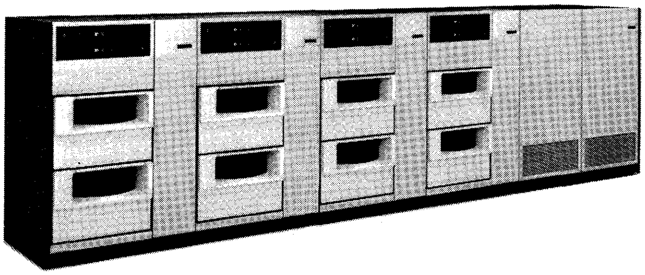


Figure 1—For computer installations in the '70s, the central processor will perform arithmetic and logic operations at speeds measured in billionths of a second, and it must draw upon vast amounts of data stored in magnetic disk units. The IBM 3330 disk storage holds 200,000,000 bytes of information and through its associated control unit, the IBM 3830, feeds data to a processor at a rate of 806,000 bytes per second

nologies now emerging will be briefly assessed. The resulting predictions are, of course, individual and not corporate, but hopefully the ideas have been drawn more from objective observations and published fact than from personal bias.

HARDWARE CAPABILITIES

While a disk file is an intricate piece of machinery, its basic capability rests upon a few key components in addition to the storage disk. The magnetic transducers record and read magnetic patterns representing the information. An actuator moves the transducers over a large number of tracks of recorded information, permitting the economy of access to a large volume of information. The switching circuits enable many transducers to share drivers and sense amplifiers for write and read operations and to share a control unit for processing signals and interfacing the disk file with a data processing system.

In analyzing the cost and performance of a disk file, areal storage density is the key parameter to consider. This stems from the fact that disk files have an inexpensive storage medium but expensive auxiliary components, and only yield a low cost per bit when the auxiliary components are shared by as many bits as possible. Higher areal density also improves performance parameters such as the data rate, access time and amount of data per access.

Along with capacity increase, disk storage devices have improved steadily in reliability and versatility. The successive generations of disk files and their control units have acquired capabilities such as on-line diagnostics, independent device servicing, rotational position sensing and command retry.

Nonetheless, the trend toward a lower and lower cost (e.g., a decrease in rental from $23/megabyte/month for the 2314 facility in 1967 to $8/megabyte/month for the 3330 disk facility in 1971) continues unimpeded because the advantages of density improvements far offset the costs associated with better component and device performance. However, density improvements will not readily enable us to produce small capacity files at significantly lower cost per byte because the cost of auxiliary hardware cannot be spread over a large enough volume of data.

To understand what is desirable and possible for future disk development, we must examine the critical parameters that control areal density—the product of linear density (bits per inch along a track) and track density.

Let us turn first to linear density which, in magnetic terms, is the number of flux changes per inch. This factor is primarily controlled by geometrically related parameters. That is, the head gap determines the minimum bit dimension that can be resolved, and the storage medium thickness and head width determine the maximum available flux. Also, our ability to keep the head gap in close proximity with the medium determines the efficiency of magnetic coupling between the two and insures better resolution by minimizing the interaction of magnetic fields from adjacent flux transitions.

In track density the limiting factor is our ability to accurately position the read-write head within a fraction of the track width. An error in positioning can cause either erasure of an adjacent track or failure to erase completely a previously written record. On readback, either case will reduce the maximum possible signal-to-noise ratio (S/N).

The common design practice is to maintain a position tolerance within 20 percent of the track width.* Early systems used mechanical detents. In the newer machines, the recording head is positioned either by a signal read from one of the recording surfaces as in the IBM 3330 drive or from an external servo reference as in the Information Storage Systems 715 drive and similar disk devices. These techniques have permitted the use of densities reaching 200 tracks per inch.

To improve the positioning tolerance in future systems will probably require servo techniques that actually use the data track itself to provide an appropriate error signal. This perhaps could be accomplished by including appropriate servo data in the basic recording code or by interspersed patterns along the track, at the price of a sacrifice in linear density.

* Because of the many individual tolerances, the actual allowable misregistrations for a particular machine is normally expressed in statistical terms.

On balance, linear density appears to be more amenable to improvement than track density. Increased linear density when compensated by reduced head-to-medium spacing could maintain the same signal, while increased track density would always be accompanied by a loss of signal for a given number of turns. Moreover, as the track is decreased below the present .005-inch width, significant problems in the accuracy of both static and dynamic mechanical positioning of the head will be encountered because of limitations in servo bandwidth, spindle bearing noise, thermal gradients, and similar mechanical factors.

Now, let us examine in more detail the components affecting linear density.

Recording medium

A particulate coating is commonly used for the recording medium. Magnetic particles dispersed in a binder have proved to be tolerant to impact and wear. This system allows us to separate the problem of obtaining the desired magnetic particle characteristics from the problem of obtaining the desired mechanical durability, smoothness, and uniformity in the binder. Consistent with the improved resolution of the recording system, the storage medium thickness has been reduced in successive improvements from 1000 microinches in the earliest disk files to below 100 microinches at present.

While some further improvement is possible, we appear to be reaching the limit of our ability to form a uniform thin coating with adequate loading and dispersion of particles of the proper magnetic materials. Further density increase may require a medium thickness below 20 microinches. The simultaneous requirements of such a thin medium for efficient magnetic coupling and of high magnetization for maintaining an adequate signal seem to be better answered by continuous magnetic films.[1] However, in such media the problems of producing desirable magnetic properties and a durable physical surface must be solved simultaneously. Magnetically, they must provide stable storage, adequate signal and low surface noise. Mechanically, they must withstand the wear from intermittent contact between the disk and the head. From evidence to date, a considerable development effort will be required to achieve a technology as consistent and reliable as current particulate coatings.

Read-write transducers

With write and read transducers, the current practice is to use a ferrite ring head with a glass gap. We can obtain an accurate and well-defined gap with a 40 to 80-microinch length, a 400 to 1000-microinch height, and magnetic pole pieces infinitely long relative to the gap length. Long pole pieces are a consequence of fabrication techniques to obtain flatness economically. Accurate control of gap height is essential since it provides a shunt path for the magnetomotive force and sense flux, respectively, during write and read.

When compared to gap, spacing, and track width, the head as a whole is a bulky structure. While smaller ferrite structures can be produced, current machining practices are a serious limitation to significant size reduction.

One of the fallouts from magnetic film memory technology is a fabrication technique which could be used to form miniaturized magnetic read-write heads.[2] The technique allows us to deposit magnetic-conductor-insulator multilayers and to shape miniaturized structures by photolithography and various etching processes, making it possible to produce narrow gaps for higher linear resolution and narrow width for high track resolution. Also inherent in the fabrication process is the ability to produce economically a multiplicity of elements. They could either be sliced from a large wafer into many chips for individual use or used as a group to provide an array of heads for fixed head files.

Structures of greater complexity than that required for head arrays have been deposited directly onto a silicon wafer in the fabrication of thin film magnetic memories.[3] This suggests the possibility of an array of magnetic heads optimally packaged with the semiconductor selection matrix and sense amplifiers on the same silicon chip. Such heads have quiescent magnetization parallel to the disk surface to avoid undesirable disturbance, high magnetization for an efficient write field, and high permeability at high frequency for efficient coupling during reading.

Head-to-medium spacing

The mechanical access device in the most recent disk files consists of a servo-regulated actuator which moves an air-bearing slider that, in turn, carries the read-write heads. The key design consideration here is the control of head-to-medium spacing.

Smooth, flat surfaces for the head carrier and the medium are essential to the control of head-to-medium spacing. Machining, polishing, and buffing techniques have been developed to consistently produce surfaces with roughness within an arithmetic average of a few microinches. Smooth surfaces prevent contact and ensure operation in a state where there is hydrodynamic lubrication by the air film carried at the boundary layer of the disk.

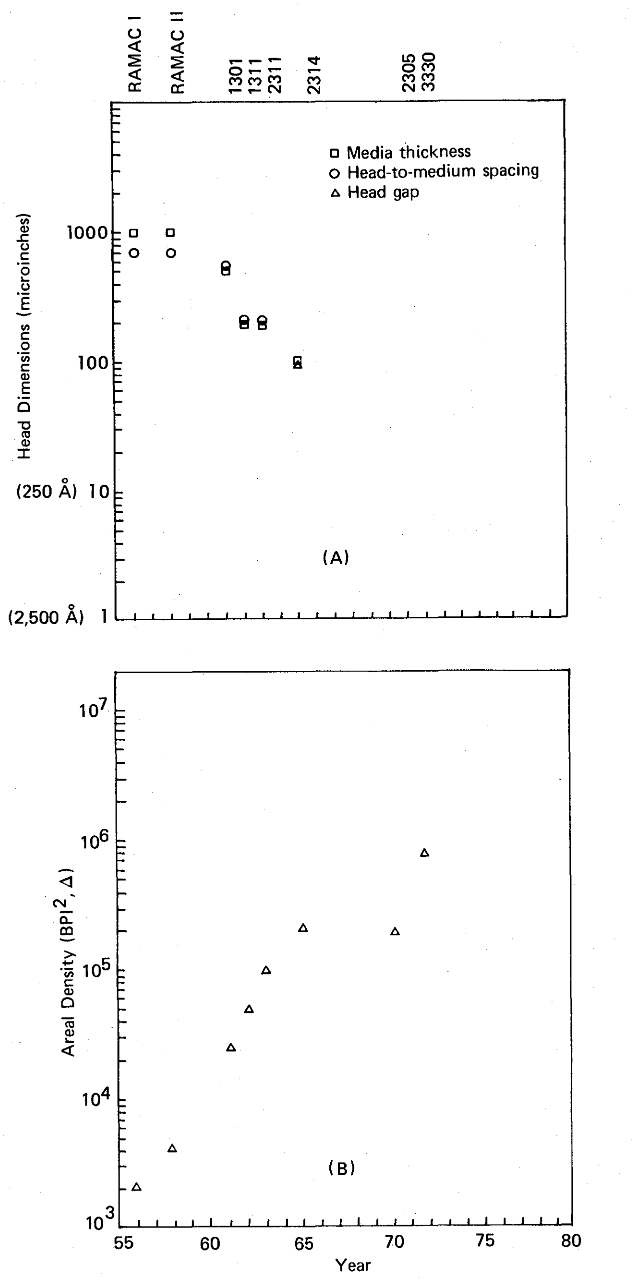


Figure 2—With successive models of disk files, (A) the head gap, medium thickness and head-to-medium spacing continued to decrease, (B) to achieve higher areal storage density

High resolution in future disk files will require considerable reduction in head-to-medium spacing. The air flow will still retain the properties of a continuous fluid medium apart from the regions near the surface where slip flow will occur. The validity of continuous flow will be a function of the Knudsen number, which is the ratio of the mean free path of the air to the spacing. As long as this ratio is less than unity, the continuous flow assumption will hold and hydrodynamic lubrication will not present any limitation. Since the mean free path of air is 2.5 microinches at atmospheric pressure, it is estimated that no hydrodynamic limitation will be encountered at spacings above five microinches.

Reduction of the spacing will also require a proportional reduction in the area of the bearings used to support the magnetic transducer. Such a reduction in area would be accompanied by much lower loads than those needed for air bearings at present.

The preceding discussion alludes to a general relationship between recording density and geometrical parameters of components. Indeed, as shown in Figure 2, successive generations of magnetic disks indicate that areal density has been increased by scaling down three key geometrical parameters: head gap, medium-to-head spacing, and medium thickness.

Reading, which is the inductive sensing of the recorded magnetic pattern in a disk, is essentially a linear phenomenon. If all geometrical parameters in the three-dimensional space are scaled up or down, while the magnetic material (specifically, the magnetization) is unchanged, the field distribution and field magnitude do not change.[4] Since the signal voltage is directly proportional to flux ([field] [length]^2) and inversely proportional to time duration ([length]/[rotational speed]), the signal voltage per unit track width per turn of read head ([field]/[rotational speed]) stays constant. The signal is also directly proportional to storage medium magnetization. Typically, for an iron-oxide disk medium at a rotational speed of 3600 rpm, a signal voltage of 10 microvolts per mil of track width and per turn of read head is obtainable.

The dimensions for the key geometrical parameters at various areal densities could be extrapolated if one set of values were known. Using the IBM 2314 as a reference, at 2×10^3 bits per inch linear density (2×10^5 bits per square inch areal density), the head gap, medium thickness, and head-to-medium spacing are, respectively, 100-100-80 microinches. Extrapolating the linear density of the 2314 file (2200 bits per inch) by a factor of 10 would reduce these parameters to 10-10-8 microinches.

Obviously a 10-microinch (2500 Å) gap is not easily achievable with the conventional ferrite technology. The problem is compounded when narrow track width is also pursued for higher track density. Thin film heads offer one solution for producing such narrow gaps. The thin recording medium required would be difficult to achieve with particulate binder coatings. However, a continuous film medium could provide a film of the required thickness with the higher magnetization needed to maximize the signal output. Typical saturation flux density for cobalt nickel film is 10,000 gauss, while that for iron-oxide disks is 450 gauss.[1]

This analysis demonstrates that the per-turn per-unit track width signal can be maintained or even improved at high linear density if we are willing to pay the price of implementing exacting geometrical dimensions and developing new materials and fabrication techniques. It should be emphasized, however, that the total signal is likely to decrease by a factor of 10 with the projected higher areal density since a miniaturized head can only accommodate fewer turns of windings than before, and higher track density requires narrower head width.

For increased areal density in the near future, better detection circuits are needed in addition to improvements in recording components and mechanical tolerances. Further density improvement will be accompanied by the signal decreasing toward the intrinsic noise levels of both the storage media and the electronic devices.

Signal-to-noise ratio

In viewing the disk file as a transmission channel, we encounter many sources of degradation in the optimal available signal-to-noise (S/N) ratio. Track misregistration detracts from the written signal and adds unerased old recording to the noise. Head-to-disk-spacing variation affects the pulse crowding phenomena. Lack of uniformity in the distribution (size and orientation) of magnetic particles as well as foreign inclusions and voids in the binder system cause extra and missing bits. Up to now, these noises have been the principal sources of error, and most of the engineering design and process quality-control efforts have been aimed at avoiding and minimizing them.

However, with the high degree of miniaturization, the intrinsic random noises will assume a significant role. They include the so-called storage medium "shot" noises and thermal noises associated with the input impedances of the sense amplifiers and the winding resistances of the read heads. The signal voltage induced in the read head is composed of the individual voltages produced by the discrete magnetic particles embedded in the binder. Their discreteness and inhomogeneity result in a random noise which is analogous to those associated with the emission of discrete electrons in vacuum tubes.

While the system noises arising from mechanical misregistration or environmental pickups can be designed to an acceptably low level, the intrinsic random noises cannot be prevented. We can only insist on an adequate S/N to ensure a sufficiently low probability of failure or sufficiently long mean time between failures (MTBF).

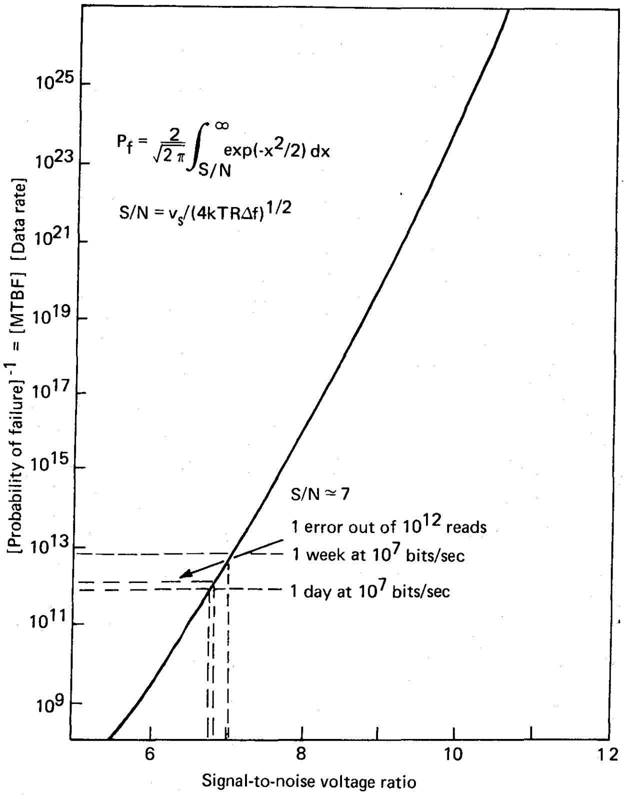


Figure 3—Mean-time-between failures as a function of signal-to-noise voltage ratio

A very striking feature of the intrinsic noises is the extremely nonlinear relationship between the MTBF and the S/N. For example, when the S/N is reduced from 10 to 5 (a factor of 2), the MTBF suffers a reduction by 16 orders of magnitude! Thus, it is imperative to have an ample margin for the S/N in order to safeguard a satisfactory MTBF or to ensure an acceptably low error rate.

To achieve an error rate of one error out of 10^{12} reads, an S/N of 7 is required (Figure 3). A signal of a few tens of microvolts per mil turn at linear densities between 10 and 20 kilobits per inch can be obtained by using improved magnetic components. If a 10-turn read head and .002-inch track width are assumed, the signal at the preamplifier is a few hundreds microvolt. Presently, preamplifiers have a thermal noise of about 50 microvolts at a frequently bandwidth of 15 MHz at room temperature. The noise voltage (V = square root of $4kTR\Delta f$) as seen by the sense amplifier is a function of the amplifier bandwidth. As increased data rate (resulting from higher linear density) requires wider bandwidth, the noise increases. For example, the same intrinsic noise source which gives 50 microvolts at 15-MHz bandwidth would give 130 microvolts at 100-MHz bandwidth. Obviously, the present preamplifiers are inadequate for the future high density magnetic disk recording. To maintain the S/N at 7, the thermal noise should not exceed 30 to 70 microvolts at a bandwidth of 100 MHz.

The typical medium shot noise from a state-of-the-art particulate disk is a few microvolts per mil turn. For a 10-turn head, the medium noise is amplified to tens of microvolts per mil at the input of the sense amplifier. To reduce the shot noises in particulate media, particles must be made smaller and more homogeneous. Fine polycrystalline continuous thin film media may also be helpful.

The thermal noises can also be lowered by reducing sense amplifier input impedance. Moreover, since the errors due to random noises are transient and recoverable, a track can be reread to correct the errors, provided that reread occurs infrequently.

Coding

So far, we have implicitly equated bits per inch with flux reversals per inch. This implies a data coding that requires one bit per flux reversal, but codes are possible which achieve greater than one bit per flux change. Current products use coding schemes (NRZI, FM, MFM) that range from one to two flux changes per bit. The choice of such codes is usually dictated by a balance between spatial resolution and timing tolerance of clocking and data detection. Codes that offer as much as two bits per flux change require clocking capability which must maintain synchronization within a small fraction of a bit time over many bit periods. Extremely narrow pulses also are needed to minimize peak shift caused by adjacent bit interference. At a 10-MHz data rate, this requires timings in the nanosecond range and is difficult to achieve economically in normal commercial environments, due to variations in disk speeds, temperature, and other factors.

This tradeoff, compounded with the extreme nonlinear error dependence on the overall system S/N makes it unlikely that practical recording devices will greatly improve upon a one-to-one relation between flux reversals and bits stored.

DEVICE CHARACTERISTICS

So far, only factors most relevant to increasing areal density have been examined. Let us now consider device characteristics which include actuator access time, latency time, data rate and volume of data per access.

The access time is limited by the mechanical design of the actuator and the servomechanisms. The latency time is half the period of rotation. The data rate is the product of linear density, track length and angular velocity. The volume of data per access is the product of tracks per cylinder (number of heads) and bits per track.

Thus, an increase in linear density improves two of the four performance parameters since it increases both the data rate and the volume of data per access. However, these improvements have an upper limit. We know that the linear density is increased at the sacrifice of S/N due to signal decrease as well as bandwidth increase. This leads to increased error rate (or probability for failure). The MTBF suffers both from the data rate increase and failure probability increase. Since the thermal and shot noises are random, the errors will be transient and recoverable; we could and would attempt occasional reread. However, when the MTBF approaches the period of rotation, continuous reread will be needed and we have reached an untenable situation.

In addition to density improvement, there are other avenues to performance improvement. It has been demonstrated in fixed head files that the access time can be eliminated. In one fixed head file (Burroughs Model 9375-3), 12,420 heads are used to serve 40 fixed data surfaces. Currently, such files are significantly more costly than moving head files of the same capacity, but again the application of thin film technology could substantially reduce this premium, at least at modest capacities.

Magnetic thin-film head arrays are composed of fewer layers of materials[2] and a smaller number of magnetic elements (typically tens of heads) compared to a film memory array (typically 10^3 to 10^5 bytes). Thus, the existing fabrication techniques as advanced by the film memory development, together with the necessary improvements in the disk surface and a mechanical slider to hold an array of heads, could produce integrated head arrays. When combined with a semiconductor selection matrix, an economical solution to reduce access time substantially may well be possible.

The latency time seems to be a different matter. Each track contains more than 10^5 bits at present, which will increase to more than 10^6 bits in the future. To be competitive with electronic random access memories which can directly access many small blocks (hundreds to thousands of bits), thousands of integrated head arrays with associated sliders and selection matrices would be needed. Obviously, this would be an inferior answer for applications requiring short latency time, and we can expect disks to continue to serve optimally as storage for relatively long records. At its best, the disk file may approach the performance of a shift register electronic memory, but it will still fall short of the performance of an electronic random access memory.

Improvement of the cost per bit through capacity increase need not be accompanied by high data rates entailing expensive electronics. By lowering the rotational speed, the economy of large volume can be obtained while holding the data rate constant. However, this would result in a proportionate increase in latency, offsetting the current effort to decrease access time and improve system throughput.

In summary, over the next decade, there is the potential to improve linear density by a factor of 5 to 10, and the track density by a factor of 2 to 4. Beyond that, the cell sizes will be limited by intrinsic noises, and the cost of further improvement would outweigh its gains.

To achieve these improvements, new technologies will have to become practical. Reaching these goals will require significant changes in basic processes, the resources of a vigorous competitive industry, and probably will take most of the decade to evolve.

ALTERNATIVE TECHNOLOGIES

The continual demand for more on-line bulk storage has attracted new technologies to contend for the magnetic disk market. While these technologies are based on interesting physical phenomena and enthusiastic predictions, their merit as future products must be measured in terms of both economical viability and technical capability.

The economical assessment of any totally new technology should cover the development cost for both hardware and programming, and address whether the possible return will justify the investment. The latter will be influenced by the range of products to which the new technology can be applied. The assessment of hardware costs should include not only the development of new storage materials, but associated component technologies and system interface. Programming costs would include the impact of both system and user programs. Most difficult of all, if basic changes in current system architecture are required, the burden of justifying a total system development falls on the new technology.

It is useful to summarize the advantages and disadvantages of magnetic disk systems with a view toward the difficulties and opportunities for introducing alternative technologies into the market place. The reversibility of magnetic recording permits updating records in place, and its remanence ensures nonvolatile storage. Concurrent development of systems configuration and disk files has resulted in considerable interdependence of operating systems and disk processing. This and heavy user investment in disk-oriented programming will discourage dramatic departure from current hardware concepts. Off-line storage capability gives the user unlimited private programs and data sets, along with the flexibility to use different computing systems. As an on-line storage device, the disk drive offers large data volume per access, has 10-millisecond access time, provides very fast data rates, and includes considerable logic and system capabilities through the channel and control unit electronics. At present, the disk file offers the largest on-line direct-access storage capacity (10^6 to 10^8 bytes, and growing beyond 10^9 bytes) at the lowest cost per bit. The technology on which such files are based is still capable of more than 10 times improvement in storage density, and hence commensurate improvement in both capacity and cost per bit.

Magnetic disk technology does have basic limitations. Unlike solid state memories, its cost advantage cannot be as easily extended to low capacity storage. It does not appear that substantially reducing latency by faster rotation or more transducers will be economical. Magnetic disk recording does not permit local mixture of memory and logic as does semiconductor technology. Finally, due to its mechanical nature, the disk file consumes space and power and requires maintenance. While these limitations do not detract significantly from the merits of the disk file for the majority of its present bulk-storage applications, they do offer opportunities for new technologies in special applications such as in space, or in minicomputers, and in computers with new architecture and operating systems.

Concentrating on bulk storage, we limit our comments to three leading alternative technologies: beam-addressable files,[5] LSI semiconductor arrays[6] and magnetic bubbles.[7]

Beam-addressable files use the speed and flexibility of optical techniques which rapidly move a light beam to access the stored data rather than mechanically moving the read-write device to the storage media as required in current disk storage. However, the problem faced by optical accessing methods is finding materials that are sufficiently sensitive to allow writing with very low energy levels, and which, in turn, provide enough energy to allow high speed readout with an acceptable S/N.

Photographic materials can satisfy both requirements. They have been used successfully, for example, in the trillion-bit memory of the IBM 1360 Photo-digital Storage System. Designed for specialized applications. the system's nonreversible film memory would seriously inhibit its extension to a more dynamic on-line environment. Systems which employ high energy lasers to vaporize holes in metallic coatings and other similar technologies face the same problems.

The most commonly proposed reversible beam-addressed file technologies are based on thermal writing and magneto-optic reading by a steerable laser beam. This approach, configured as a disk file, is attractive in concept. It shares with conventional disk equipment the simplicity of a homogeneous, stable, but reversible storage medium. At the same time, the beam technology offers the possibility of eliminating mechanical accessing mechanisms.

In practice, there are serious drawbacks. The magneto-optical read and thermal write transducers require the development of a storage medium with large magneto-optic effects. Although such materials exist (e.g., MnBi for room temperature operation, EuO or Fe-doped EuO for low temperature operation), they are by no means optimized nor do they have an established materials technology comparable to particulate oxide media or permalloy magnetic films.

Moreover, at present, nonmechanical deflectors can only steer a light beam to a limited number of recording tracks, and a simple solution to the technical problems involved does not appear to be immediately at hand. An alternative to this scheme utilizes mirror rotation in combination with a matrix-controlled laser array to select a track, and mechanical rotation to scan the track. Another drawback is that the dc bias field cannot be easily switched on and off, thus necessitating two rotations to effect rewrite.

However, the projected storage density (and thus, roughly cost per bit) of a magneto-optic file is comparable, but not significantly superior, to that for conventional magnetic recording. This factor, along with the current state of materials development and the mechanical-accessing feature suggests that the magneto-optical file is not likely to replace conventional magnetic files unless the latter fail to reach their projected improvement in areal density.

The semiconductor industry is well established and broad-based. Semiconductor components have found their applications in the memory hierarchy, ranging from the CPU to cache and main memory. Developments in FET devices and charge-coupled devices have further heightened the hope that they might become sufficiently cheap to replace disk files.

Since cost is the central issue, let us consider if there is a tangible technical basis for predicting whether semiconductor costs will approach magnetic disk costs. Basically, magnetic disks are only expected to perform the storage function, while the semiconductor storage cell must enable selection and transmission as well. Consequently, semiconductor devices have much more stringent materials requirements, and many elements have to be electrically connected.

The high quality single crystals, the high-resolution photolithography, and the many processing steps which tend to compound failures are not likely to lead to a cost even close to that of the disk file for bulk storage. However, it is quite reasonable to expect that solid state memories may eventually catch up with the low-capacity end of the disk files, namely the 10^6 to 10^7-byte fixed-head files. The cost per byte at that capacity level, whether for semiconductors or magnetic disks, is at least 100 times higher than that for the large-capacity disk files.

The magnetic bubble technology has shown impressive progress. It appears to have found materials for high-density storage, thin film techniques to facilitate fabrication, and processing steps similar to those well established for the semiconductor technology. But as far as cost is concerned, the comments concerning semiconductors apply equally well to the magnetic bubble technology. Perhaps the simpler structure, resulting in simpler processing steps, will give magnetic bubbles a cost advantage over semiconductors, but this still should be far above the cost of large disk files.

Undoubtedly, there are some intrinsic properties which could be explored for significantly new file configurations. For example, magnetic bubbles exhibit capabilities for both logic and nonvolatile storage. The logic capability enables on-chip decoding and amplification, and offers the potential of removing the demarcation line between storage unit and control unit. The nonvolatile storage capability allows both on-line and off-line stable storage.

In conclusion, the magneto-optical file is configured for bulk storage, but it still lacks satisfactory storage media and accessing devices. The photo-digital file (non-reversible media) implies a drastically different programming system and architecture. The semiconductor and magnetic-bubble memories do have a sound technology foundation, but their present configurations and cost projection are only suitable for large memories (10^6 to 10^7 bytes), but not for bulk storage (10^7 to 10^9 bytes).

PAST AND FUTURE HIERARCHIES

Having explored what technical capability can be expected, let us switch the frame of reference to that of a systems designer and look at what will be desirable and useful in the future development of disk storage.

In the past, storage has consisted of high-speed semiconductor memories, lower speed ferrite cores, fixed head disks or drums, movable head disk files and magnetic tape units.

As a sweeping generalization, to a system designer any form of hierarchy of storage is an undesirable compromise. Where possible, all the memory associated with a data processing system should be available in a single configuration with an access time comparable to the logical processing speed of the system. While in practice this is economically unrealizable, the fewer the levels of storage the better because each level produces a new set of problems and a new set of complexities.

In the past when the user worked with relatively simple problems that were well structured, stable and few in number, he could be expected to deal with the problem of knowing where his data was stored. He was also acquainted with something of the characteristics of the devices that were being used so that he could adapt his program to make efficient use of these devices.

Having user programs manage storage in such a manner implies that no significant change in the system can occur without impacting both user and systems programs. However, the life of an applications program needs to be easily extendable across several generations of hardware and operating systems.

Further in the future, there will be a multiplicity of users with less and less sophistication, as well as stored data sets for common usage whose characteristics the users do not understand, or even wish to understand, and cannot predict. Therefore, programming operating systems have evolved which perform a data management function and make the hierarchy of storage devices "transparent" to the users. This process started with the very primitive I/O subroutines of the '50s and proliferated in the mid-60s to include a wide variety of access methods and supervisory scheduling functions. These now are beginning to include virtual memory techniques and entire data handling subsystems such as Honeywell's Integrated Data Store and IBM's Information Management System. In addition, the increasing importance of shared data and the necessary associated security procedures will require new centralized storage management functions.

On the other hand, the management of bulk on-line storage has always been a critical issue in determining systems performance. Although substantial success has been achieved in transparently managing data movement between semiconductor buffers and main memory, similar techniques applied to data transfer between main memory and disk storage have achieved only limited success. The problem is that efficient management of today's on-line and off-line storage demands considerable attention to block size, sequentiality and patterns of periodic access. And the direct extension of these techniques to still another level of bulk storage is far from obvious as there is gap in storage management procedures between what we need to do and what we know how to do.

With this background, let us consider the hierarchy of storage defined previously and the impact of evolving technology on at least its exterior configuration.

Over the next 10 years, two main factors in addition to improvements in disk technology will be evident in computer storage development. First, there will be a substantial cost reduction in semiconductor memory or an alternative such as magnetic bubbles. Neither, however, is very likely to approach disks on a cost-per-bit basis. Second, extremely large capacity on-line magnetic tape devices will become available, such as the Ampex Terabit Memory System or other systems of mechanically accessed tape reels. Although we will see increasing variation in the possible characteristics of such large capacity storage devices, they will probably all characteristically have a lower cost per bit than disks during this decade and substantially poorer random access capabilities.

The cost reduction in semiconductor memory, or its equivalent, may result in its supplanting ferrite cores, fixed head disks and drums by the 1980s. Moreover, as these large capacity memories commonly reach 10^6 to 10^8 bytes, more jobs will be processed concurrently and more of the data required for a single job will be held in main memory. In this case, the present access time to a disk store can be tolerated, and pressure to improve this parameter will diminish in favor of an emphasis on cost, capacity, reliability and packaging advantages.

A more interesting question is the relationship of tapes and disks in future on-line storage systems.

A tape-oriented system might be envisioned as a three-level hierarchy, consisting of a very large main memory (10^7 to 10^8 bytes), a moderate-sized disk system, plus a tape based mass store.

In using this system we are confronted with the significantly increased difficulty of managing such a hierarchy at exactly the time when system, rather than user, management of storage is becoming more and more important. Looking up an entry in last year's ledger should not cause any substantial portion of that ledger to be staged to disk. On the other hand, because of the slow random access rate possible with the tape-based device, it is clear that, when a data set becomes active, independent requests to that data set must be limited to a few, if anything like the efficiency of current systems is to be maintained.

Such tape-based devices could reduce the need for capacity in disk storage. However, maintaining an equivalent rate of systems activity would substantially increase the requirement for accesses to each unit of disk storage since a smaller number of drives must handle all previous memory disk transfers plus the added accesses caused by tape-to-disk transfers. Thus, the cost savings in the disk component of the system may be less than expected.

As a result, it appears that the limitation of access rate in tape-based devices, however mechanized, will offset their lower cost unless a very low percentage of the accesses directed to the data base require access to that portion stored on tape.

These predictions do not imply that a tape-based library will not play an important role in future systems as an archival or I/O device. What they do imply is that such a device will not impede the growth of disk file capacity for on-line storage.

CONCLUSION

In summary, the picture is one of a continuing evolutionary development of present disk file technology with at least an order of magnitude increase in storage density as well as total storage volume, and a corresponding decrease in the cost per byte stored.

In a future hierarchy, the upper limits of solid state memory capacity imposed by cost considerations will move to the 10^7-byte range, either by continued development of silicon technology or the introduction of a new technology such as magnetic bubbles. However, above 10^8 bytes there still will be at least one and probably two orders of magnitude of cost-per-byte difference between solid state memories and large magnetic disk files. Thus, both will coexist.

The increased size and higher data rates of electronic memories will also tend to increase the need for blocking of records, permitting increased efficiency in the use of disk storage space. The very high data rate that will be possible and necessary as a consequence of further disk developments will make the interchangeability of disk packs more a feature that insures data availability than a typical means of moving data into and out of an active system data base.

Therefore, the role of disk storage will become more and more the maintenance of current on-line data with lower performance tape libraries performing an archival function.

Given the momentum of magnetic disk technology and the availability of new materials and fabrication technologies to provide a significant increase in density, it is unlikely that any other technology will displace magnetic recording in the coming decade as a principal data storage media for the industry. For some other technology to displace it, probably the alternative method would have to provide functions we do not foresee rather than merely a more economical or efficient means of storage.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the helpful comments by R. Arnold, B. Beers, L. Comstock, F. Kurzweil, G. Santana and C. D. Mee.

REFERENCES

1 F E LUBORSKY R O McCARY
Magnetic film materials
Proc IEEE 59 pp 1474-1480 1971

2 L T ROMANKIW I M CROLL M HATZAKIS
Batch-fabricated thin-film magnetic recording heads
IEEE Trans on Mag 6 pp 597-601 1971

3 H CHANG N J MAZZEO L T ROMANKIW
0.25×10^6 Bits/In^2 NDRO coupled film memory elements
IEEE Trans on Mag 6 pp 774-778 1970

4 E P VALSTYN R I POTTER A PATON
Validity of scale models in magnetics
Computer Design 10 pp 94-96 1971

5 G J FAN
Magnetooptic storage
IEEE Trans on Mag 7 pp 590-594 1971

6 L M TERMAN
FET memory systems
IEEE Trans on Mag 6 pp 584-589 1971

7 A H BOBECK H E D SCOVIL
Magnetic bubbles
Sci Am 224 pp 78-91 1971

8 W A GROSS
Gas film lubrication
Wiley New York 1962

9 H BLATT
Random noise considerations in the design of magnetic film sense amplifiers
MIT Lincoln Lab Group Report 1964-6 August 17 1964

10 C D MEE
The physics of magnetic recording
North-Holland Publ Co 1964

11 B K MIDDLETON
The dependence of recording characteristics of thin metal tapes on their magnetic properties and on the replay head
IEEE Trans on Mag 2 pp 225-229 1966

12 A S HOAGLAND
Mass storage revised
Proc FJCC 1967

13 F D RISKO
New horizons for magnetic bulk storage devices
Proc FJCC pp 1361-1367 1968

14 G BATE J K ALSTAD
A critical review of magnetic recording materials
IEEE Trans on Mag 5 pp 821-839 1969

15 V N CONSTANTINESCU
Gas lubrication
Am Soc Mech Engs 1969

16 J C MALLINSON
Maximum signal-to-noise ratio of a tape recorder
IEEE Trans on Mag 5 pp 182-186 1969

17 H HOWARD
Memories—Modern-day musical chairs
Part I EDN/EEE July 19-28 1971

18 A PATON
An analysis of the efficiency of thin film magnetic recording heads
Submitted to J Appl Phys

19 E P VALSTYN
Integrated head development
Conferences on Advances in Magnetic Recording January 1971 and Annals NY Acad Sci to be published

20 E W PUGH
Gaps, cliffs, and trends in storage hierarchies
To appear in IEEE Trans on Mag Dec 1971

21 G G SCARROTT
The storage complex considered as a system component
To appear in IEEE Trans on Mag Dec 1971

22 M L WILLIAMS R L COMSTOCK
An analytical model of the write process in digital magnetic recording
To be published in the Proceedings of the 17th Conference on Magnetism and Magnetic Materials 1972

Ultra-large storage systems using flexible media, past, present and future

by WILLIAM A. GROSS

Ampex Corporation
Redwood City, California

INTRODUCTION

This paper aims to survey past and present, and predict possible future ultra-large digital memory storage systems It is easy to describe what exists, but difficult to present meaningful predictions. To cope with this objective, characteristics of systems that *can* be available in 1976 and 1981 will be described and explained. Since it is probable that not all of these will be available, the necessary and sufficient conditions have to be identified. These conditions are qualitatively reviewed by discussing the development process of systems such as ultra-large storage systems.

The functional need for ultra-large storage systems has arisen because the power of computers, the complexity and amount of processing done, and the back-up memory sizes have continued to grow. Each year more data processing managers find their tape libraries and stables of disc files getting out of hand. An on-line, but slower access memory is required to fill this new memory hierarchy niche.

In this paper the characteristics of ultra-large storage systems and the boundary between such systems and large storage systems are defined. After reviewing the development process, identifying physical, engineering, software, and system implication, technological considerations relevant to ultra-large storage systems are detailed. Then possible bit-by-bit and holographic storage systems are described and realizable properties predicted. The sufficient condition required for existence of these systems is funding of the development and production. There are too many vagaries in patterns to predict this. The paper does aim to set the stage for rational decision making. Decisions yet to be made will determine which of the possible systems will be produced, and when.

DEFINITION

An ultra-large digital storage system is defined by function and size. It fills a new position in the memory hierarchy of large computer systems, by replacing off-line data stores, and placing the entire back-up storage on-line. Since ultra-large storage systems back up tape transports and disc files, the size will vary with time. In 1972 ultra-large digital storage systems place on-line 10^{11}-10^{12} bits with access time to any data set in approximately 1-10 sec. In ten years the largest size used is likely to approach 10^{14} bits. The lower bound of memory size will be about three to ten times the storage capacity of the largest commonly used disc file units (or competitive storage systems such as may exist if large bubble memories become practical).

Ultra-large memories are systems in their own right and can be operated in stand-alone modes. More than one CPU can be connected to an ultra-large storage system without changing any of the computers' internal software operating systems. Successful systems will have both hard and soft wiring, a minimum of moving parts, and be very reliable. Flexibilities should permit simultaneous writing, erasing, reading and searching. Though data can be stored off-line for archival purposes, it appears likely that operating data will be kept on-line. The high packing density (up to three orders of magnitude fewer reels required) and low shelf storage cost (about 10^{-6} cents per bit) may induce the user to duplicate some data for shelf archives.

An assessment of existing computer systems, and trends leads to specifications of ultra-large storage system performance objectives today, and in five and ten years. The smallest sizes of ultra-large storage systems, $5(10^{11})$ bits in 1971, 10^{12} bits in 1976, and $5(10^{12})$ bits in 1981 are likely to have to sell for about

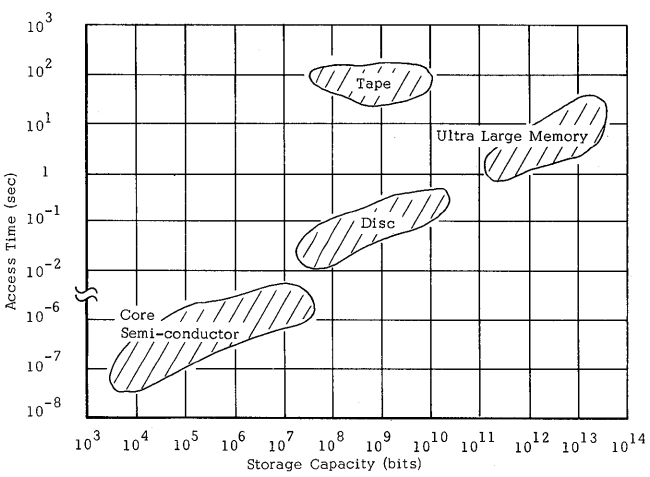


Figure 1—Storage capacity vs. access time for memories in the computer system hierarchy

one million dollars each. The largest size should be about ten times larger, and the cost less than three times as much as for the smallest system. Data accuracy (uncorrectable readout error rate) should be at least 10^{-10} in 1971, $5(10^{-11})$ in 1976, and 10^{-11} in 1976. Throughput will have to be at least 5 Mbps, i.e., $5(10^6)$ bits per second, in 1971, 10 Mbps in 1976, and 15 Mbps in 1981. Because of the mechanical access required, average access is likely to be about fifteen seconds throughout this period. Cost per bit off-line should be less than 10^{-6} cents per bit (with the storage medium fully used) throughout 1971-1981. Systems, of course, must be very reliable. Because of the size of the systems, capability of graceful degradation is useful.

These characteristics, when plotted on cost/size/access time charts as in Figures 1 and 2, clearly indicate the hierarchical position with respect to other memories.

Functionally, writing and storage of vast amounts of data, like 10^{12} bits, and accurate readout for very low costs appear to require storage in a volume. I believe that the system objectives described cannot be met in this time frame by storage in a solid volume; therefore, it is necessary to build up a volume, sheet by sheet, as in a scroll, or book. Hence, technologies reviewed here, relate only to recording on flexible media which may be stored on reels or in bins.

Assuming data accuracy, and retrieval times to be satisfactory,* storage of 10^{12} bits at a data density of 10^6 bits per square inch (as in magnetic recording in 1971) requires 10^6 square inches, or about one sixth of an acre. To avoid substantial mechanical storage, a storage system of less than 1000 square inches would be required. For 10^{12} bits, this requires 10^9 bits per square inch. Though electron beam writing with this density is possible, and readout from small areas can be demonstrated, readout under the specifications required is not practical within this time frame, and, in fact, may not be achievable. Ultra-large storage systems are therefore likely to require mechanical movement and accessing of substantial areas of flexible media.

BOUNDARY BETWEEN LARGE AND ULTRA-LARGE STORAGE SYSTEMS

As observed, the boundary between ultra-large, i.e., flexible media stores, and large, i.e., disc file stores, will move with time and with system application. Harker's companion paper[1] reviews the present state of disc files, and predicts the characteristics of disc file units in ten years. Not knowing his assessment at the time of this writing (Sept., 1971), I will predict a one order of magnitude increase in disc packing density, perhaps using magneto-optic readout techniques from plated discs. In ten years, therefore, it should be practical to use disc files to store 10^{11} bits, and the boundary between large and ultra-large storage systems would move from the 10^{10}-10^{11} bit region of today to 10^{11}-10^{12} bits in 1981.

PROCESS OF ULTRA-LARGE STORAGE SYSTEM DEVELOPMENT

A viable memory storage system, or for that matter, any technological device or system must have passed

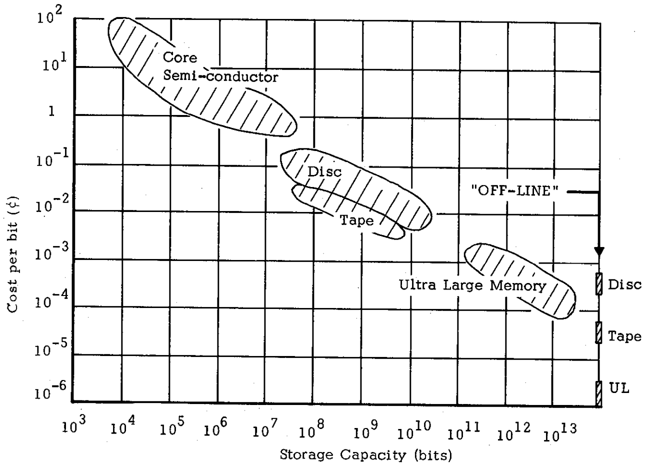


Figure 2—Storage capacity vs. cost per bit for memories in the computer system hierarchy

* It is important to note that storage density should always be specified together with data accuracy, and access times and data rates, because of the substantial tradeoffs which occur between these factors. Sometimes high density storage is specified, and the low realizable data accuracy may not be mentioned or even understood.

several necessary conditions imposed by physics, engineering hardware and software, architecture, economics, and the market (e.g., timing). Each of these factors is reviewed in this section identifying its function in the development process. Understanding them is a prerequisite to either making or reviewing predictions.

1. *Physics*: The fundamentals of physics, and materials, *including the imperfections attainable in practice*, must be such that the desired properties are achievable *with substantial margin.* For memory systems, a first step is commonly to demonstrate that sample bits of data can be recorded and their existence determined by some convenient means; techniques for practical readout being speculated. I call this step *physical feasibility*. The specific demonstration may cost from $5000 to $50,000.
2. *Engineering, Hardware and Software*: Design and repeatable production of reliable devices or systems must be economically achievable in large numbers. Operation must be economical, and occur in commonly available environments without unplanned downtime. There are many engineering implications, a few of which are worth noting.

   a. First, we must often develop processes for producing improved materials and improved fabrication methods. Different processes and methods are likely to have to be developed for each order of magnitude increase in final production.

   b. Second, both materials and designs must be such that required performance, economy, reliability and lifetime are possible. In the case of memory systems, three performance criteria must *simultaneously* be achieved with substantial margin to allow for production and usage variations: data packing density, write/readout error rate, and time considerations (data rate, access time, write/readout cycle time). Most physics feasibility demonstrations (1) involve techniques and materials in which *only* two of the above three criteria can be met. Unless there is likelihood that techniques for meeting the third criterion can be met, the study should be promptly dropped.

   c. Third, the design should approach elegance as closely as possible. By elegance is meant a sort of canonical (optimum in the sense that necessary functions are well met and superfluities are absent) design in which the machine or system is well adapted to people (the users) and in which there are the fewest number of parts, the least amount of material used, and the least amount of power required for operation.

   d. Fourth, the design should minimize both initial cost and long-term operating costs. Wide tolerances on mechanical parts made of durable materials not subject to fatigue or state change failures are desirable. There should be little, if any, contact between parts to minimize wear. Strip files have had difficulties with objectives (c) and (d). All models produced have had reliability problems; none is now being produced.

   e. An *engineering model*, costing perhaps ten to one hundred times the cost of the physics feasibility model (1) has to be built to give confidence that the engineering considerations such as (a)-(d) can be achieved.

   f. If satisfactory performance is demonstrated an *engineering prototype* needs to be built. It may cost from one hundred to one thousand times the cost of the physics feasibility model. The engineering prototype will probably include hardware, firmware, and software development as well as system studies relating to potential applications.

   g. *Software studies and development*, and *systems planning* need to be part of the engineering effort from the inception so that timely, canonical and least expensive development can be approached. Otherwise, major redesign will be required as a minimum, and more likely, too much of the design will be frozen and an optimum configuration not possible.

   h. Finally, a *manufacturing prototype*, costing approximately the same as the engineering prototype, and *documentation*, costing as much as the entire technical effort, is required to prepare for production. It has been my experience, during fifteen years in the data processing industry, that seven to ten years and seven to ten million dollars is a ballpark estimate of the cost required to move a system, such as an ultra-large storage system from conception to completion and delivery of the first production system.

3. *Architecture*: Computer architecture has changed substantially in the last fifteen years. There has been a continual growth, in computer systems, of software size, complexity, hierarchies of

memories, communications, peripherals, and corresponding decreases in many unit costs. The evolution both toward being easier to use (better adapted to people), toward greater variety, and toward increasing maximum on-line size appears likely to continue for more than the next fifteen years. It is this latter growth which establishes the growing market for ultra-large memory storage systems. To use such big memory systems, however, requires that they fit into users' information systems. Computer architecture is presently provided by OEM's. More sophisticated users are beginning to develop their own architecture, and increasingly more will in the future. Both general and special architectures will evolve during the next decade. Many of them will include ultra-large storage systems.

Design of an ultra-large storage system should be such that evolution to larger sizes over the next decade or so is possible without significant impact upon the users' operating systems: There should be negligible changes in software or system organization required. Specifically, any ultra-large storage system produced today should be such that more than order-of-magnitude increases in storage size can be accomplished through the next 5-, 10-, and 15- year intervals without requiring significant software and system changes.

4. *Economics*: The admission price for a new product has to be paid. It has been observed that this price for ultra-large storage systems is likely to run to 10 million dollars. Systems worth many times the admission price have to be sold to justify the time value of the money invested. The market estimate of those funding the development is part of the economic gate. Other parts are estimates of production costs and sales price.

   For a new ultra-large storage system to be produced by 1976, a good part of the admission price must have already been paid, and the rest of it assured.

5. *Timing*. Timing of a new product is always critical. The optimum time is bounded by *too early* (applicable technology not sufficiently developed, market need not yet developed) and by *too late* (market established and being supplied, and price reductions under way). Expecting development of an ultra-large storage system to take more than five years, *a new system can be expected in 1976 only if an engineering model has already been built.*

TECHNOLOGY CONSIDERATIONS LIKELY TO IMPACT THE NEXT DECADE

Material, electrical, mechanical, and magnetic technologies both permit construction of ultra-large storage systems, and limit their performance. The last section discussed some of the process of memory system innovation. This section lists some of the more significant technological considerations required in making predictions.

1. *Materials*: New recording technologies usually require novel storage media. In fact, invention of a new medium often triggers the first feasibility demonstration of a potential new storage process. Unfortunately, though, the quality of existing memory storage media is so good, as a result of *thousands* of man years of development, that *it is extremely difficult for any new approach to achieve the cost/performance improvement required for the resulting system to be successful.*

   The only practical ultra-large memory storage media now in use are silver halide film and magnetic particle tape. They are comparatively reliable, defect-free, and low in production cost. Other media appropriate for ultra-large storage systems have not yet proved themselves. Evolutionary improvements in both silver halide and magnetic tape will continue to be made.

   Both silver halide and magnetic tape, upon analysis, show substantial life as affirmed by National Bureau of Standards tests. After a hundred years we will be in a better position to tell how the mylar backing, which appears to be the controlling factor, holds up. There is good reason to believe that, since 10 year life has been shown, much more life than that can be expected. No storage medium has been certified to have permanent life.

   Silver halide tape used in microfilm applications has the advantage that it does not require much positional accuracy for data retrieval; both silver halide and magnetic tape, used for digital bit-by-bit storage, do. The National Archives finds that it must rewind stored computer tapes using longitudinal track recording every two years. The reason is that temperature variations of even ±5°F lead to tension changes, particularly in the inner one-third of stored tape on reels. The problem with 800 bpi tape is that the transports do not have self-clocking and can cope with very little tape stretching. (The 1600 bpi transports do have self-clocking and can cope with more stretching.)

Experience with transverse recorded magnetic tape is completely different. Transports seem easily able to servo to the two-inch long tracks even after substantial abuse including years of storage in wide temperature ranging environments. In fact, some of the earliest video tapes recorded 15 years ago can be played back with no recognizable loss in signal quality.

The National Bureau of Standards has investigated samples of storage media supplied for the hole-burning Unicon* (which will be described later). Their study revealed that resolution has been accomplished and that holes were produced without apparently damaging the mylar substrate. So long as nothing happens to the plating during the passage of time, the life, like that of silver halide and magnetic tape should be at least 10 years, and probably much more.

2. *Electrical*: Continuing improvements in cost-performance-life-complexity of integrated circuits is leading to increasing use of IC's replacing engineer-designed circuits. Furthermore, built in mini-computers permit use of soft wiring (internal computer control), instead of hard wiring (conventional circuits using conventional circuit elements), and firmware (use of ROM's within soft wired systems). The consequence is increasingly sophisticated final systems providing improvements in performance, cost, and flexibility of use. (Ironically, improvements in IC cost/performance result in corresponding improvements in all hierarchy levels of memories, including that of cores, which IC's are attempting to surpass by cost/performance improvements.)
3. *Mechanical*: Reliability problems and breakdowns seem invariably to be due to mechanical and/or material failure. Comparatively few seem to have recognized this. As has been implied, getting out information is vastly more difficult than putting it into most storage systems. In fact, my experience has led me to observe that the development cost to prove readout capability is likely to be two orders of magnitude greater than the cost to demonstrate recording. Most searches for funding for new storage techniques have to take place between these phases.

Practical reproducibility of production parts, and most importantly, accurate positioning of the readout transducer limit the performance of electro-mechanical storage systems to much lower storage capacity than physical and material considerations would permit with perfect mechanical parts and positional control. In fact, it is well-known, but sometimes ignored, that laboratory performance predicted or even achieved in feasibility demonstrations is often more than an order of magnitude better than can be reliably achieved in an operating system.

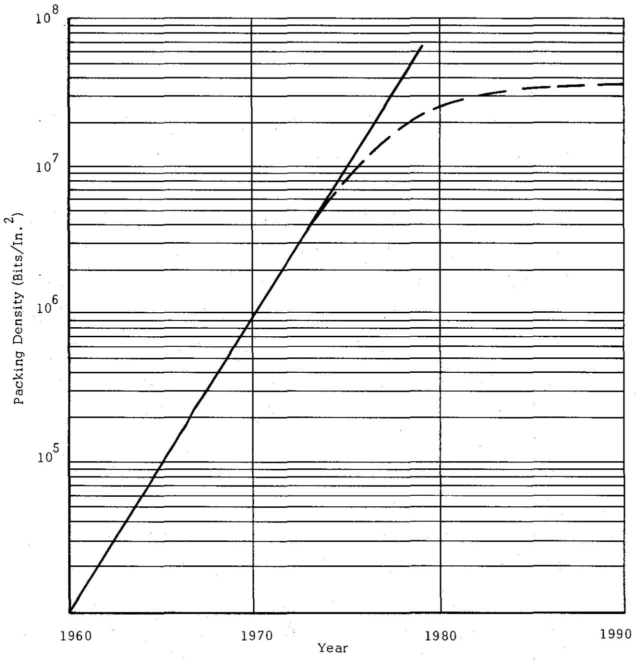


Figure 3—Area packing density, past experience and estimate of what is realizable in commercial products in the future

Conventional, longitudinally tracked magnetic tape transports are a case in point. For economic reasons, and because of the byte-across convention, tape skew has limited linear packing density to 1600 bits per inch. Instrumentation transports, however, not limited by byte requirements, record and reproduce over 25,000 bits per inch. The ultimate achievable in practice appears to be about 40,000 bpi, although the magnetic limit is about 100,000 bpi.

Packing density is limited by mechanical position reproducibility for readout, machine to machine. The limits are comparable for both flexible and rigid storage media. The storage density, attainable in mechanically accessed systems whether with rigid or flexible substrates, and regardless of the recording medium, appears to be fundamentally limited by mechanical and material considerations. Figure 3 illustrates how

* Trademark

this density has improved over the last decade, and how I believe it will level off. The figure applies to both bit-by-bit and holographic techniques.

Some[2,3,4,5,6] have used, or proposed, electron beam and laser beam systems with extremely high packing densities to overcome mechanical limitations. Figure 3 still seems to apply, but for different reasons. These will be discussed later in the section on beam access.

4. *Magnetic*: Magneto-electric effects as in core memories, and magneto-electromechanical effects as in disc files and tape transports are used for large memories today. It appears that magneto-electromechanical effects will be used for large and ultra-large storage systems for the foreseeable future. It has already been observed that magnetic tape can store information at vastly greater packing density than we will ever be able to deposit and retrieve it. The other magnetic limitation is in the transducer, the magnetic head. The inductor heads now used become lossy at high frequencies and are generally impractical for use above 15 MHz. This places an upper bound upon single channel data rates.

   There have been many efforts, such as the magneto-resistive head,[7] to produce practical non-inductive, monolithic heads. None, however, has proven to be practical to date. The appearance of a monolithic head stack which can write and read with equivalent performance, but at one-tenth the cost, would result in substantial changes in the design of large and ultra-large storage system memory elements. In the meanwhile, inductor heads will increasingly be made of hot-pressed ferrites for three reasons: superior wear, yield in production, and magnetic performance.

REVIEW AND PREDICTION OF SYSTEM PERFORMANCE

Assessing possible developments in memory technology is of major interest to many groups in addition to the technologists and marketers. Among these are the system designers, senior corporate managements, and venture capitalists. Eshenfelder,[5] Weil,[6] Hoagland,[8] and Best[9] have written papers containing useful survey assessments for ultra-large storage systems which have appeared in the last five years. The technologies and systems I will review fall into two classes: discrete bit and holographic. Discrete bit systems have been, are, and may be:

1. Magnetic-oxide magnetic tape, erasable.
2. Plated tape for magneto-optic read, erasable.
3. Photographic, silver halide tape, using electron or laser beam write and read, permanent.
4. Metallized tape, electron, or laser beam hole-burning write, and readout, permanent.

Holographic systems may be:

1. Photographic, silver halide tape, laser beam write and read, permanent.
2. Plated tape for magneto-optic read, erasable.

Each of the preceding will be reviewed, necessarily in a limited fashion. It is important to note that the predictions for 5 and 10 years from now are not predictions of what will exist, but of what is possible physically, engineering-wise, and system-wise. Whether they are developed, produced, and offered for sale depends upon whether developments are funded.

A. *Discrete Bit Systems*

1. *Magnetic-Oxide Magnetic Tape, Erasable*

Magnetic tape storage fundamentals have been so thoroughly reviewed in the literature[8,10] and somewhat here, that I will focus upon storage systems. Performance data are listed in Table I.

a. *1966*. The IBM 2321[11] Data Cell was introduced in April, 1964. It was the first significant ultra-large storage system to be placed on the market. Like preceding strip-file memories, it suffered from reliability problems. Competing disc files have superseded it. Error rate is not specified; errors are detected by the system and reported to the CPU. Access time is an approximate average.

b. *1971*. The only available magnetic-oxide ultra-large storage system in 1971 is the Ampex TBM,* or Terabit Memory.[12] (Tera is the standard designation for 10^{12}.) The system uses the transverse recording technique developed 15 years ago for video recorders, and widely used since then. Innovations include system design permitting easy growth to both larger size and to new generations without changing software, graceful degradation, and parallel channels for

* Trademark

TABLE I

		Discrete Bit, per Station															
		Mag particle tape, electromagnetic (erasable) (1)				Plated tape, Magneto-optic (erasable) (2)				Photographic Electron or laser beam, silver halide film (permanent) (3)				Metallized tape, laser vaporization (permanent) (4)			
		1966[1]	1971[2]	1976[2]	1981[2]	1966	1971	1976	1981	1966	1971[3]	1976[4]	1981[4]	1966	1971	1976	1981
a - Size	10^{12} bits	.0032	.1-3	.1-7	.1-15			1-30	1-50		1	3	10		.1-1	.1-3	.1-50
b - Raw Packing Density	10^6 bits/ in^2	.098	1.4	3	7.5	NA	NA	15	25	NA	2.8	6	10	NA	25	25	25
c - Data Rate	10^6 bps	.44	6	16	20			15	50		2.5R .55W	200	300		4	6	15
d - Access Time	sec	.45	15	15	10			15	10		7	15	10		9	9	9
e - Error Rate	10^{-10}bits,uncorrectable		.7	.5	.1			60	6		100	1	1		600	60	6
f - Cost per Bit on Line	10^{-4} cents	45	5-1	5-1	3-.3			2-.5	2-.5		1	1	.2		5-1	5-1	5-.5
g - Cost per Bit Off Line	10^{-6} cents	160	1	.5	.2			1	1		8.3	1	.5		2	1	1

		Holographic, per Station								
		Photographic, laser beam, silver halide film (permanent) (5)				Plated tape Magneto-optic (erasable) -6-				
		1966	1971	1976	1981	1966	1971	1976	1981	
a - Size	10^{12} bits				1-50				1-40	a
b - Raw Packing Density	10^6 bits/ in^2				30	NA	NA	NA	25	b
c - Data Rate	10^6 bps				50-500				50-500	c
d - Access Time	sec				15				15	d
e - Error Rate	10^{-10}bits,uncorrectable				.1				.5	e
f - Cost per Bit on Line	10^{-4} cents				1-.1				1-.1	f
g - Cost per Bit Off Line	10^{-6} cents				.2				.5	g

(1) IBM 2321 Data Cell, (f is estimated at 50 times .9(10^{-4})¢ rental per month

(2) TBM TM

(3) IBM 1360 (Electron Beam); W=write, R=read

(4) Laser Beam, estimated

(5) UNICON TM

simultaneous reading, writing, and 1000 ips search. This appears to be the only system available in the next few years which meets all of the specifications listed at the beginning of the paper. Most parts of the first system have been delivered, and customer acceptance tests are scheduled for March, 1972.

A storage system meeting some of the specifications[13] is said to be under development in IBM. It is a three-inch wide multitape loop. In the absence of specific information, I can only speculate that the system will aim to produce automated tape libraries having up to, say 10^{11} bit storage capacity. Tape loops can be stored in chambers, accessed pneumatically, then rotated at high speed for writing, reading, and erasing. Reliability problems associated with the extremely large number of rotational cycles requires, i.e., tape edge damage, and splices (if used) will have to be solved. If produced, and reliability is demonstrated, this large storage system would impact the lower bound of the ultra-large storage market. It remains to be seen whether achievement of the projected performance is possible with a reliable system.

c. *1976*. Packing density, and, therefore, total capacity of the TBM* can be doubled in five years, and the data rate increased to match increasing capability of central processors.

The tape loop, if successful, should also double capacity by about 1977. If tape loops become established, any competitive large-storage systems will have to leapfrog in cost/performance in order to be used. Ultra-large storage systems will continue to be required for larger files.

d. *1981*. By 1981 the TBM* cost/performance can increase as shown in Table I. If tape loops are successful, they would continue to meet needs at the low end of the ultra-large storage system market. Tape loop packing density is likely to remain under one million bits per square inch because of registration difficulties.

2. *Plated Tape, Laser Record, Magneto-Optic Readout, Erasable*

At this point it is useful to refer to Table II which was prepared mostly by Irving Wolf. The table shows that one material, cobalt-phosphorous is practical to be plated upon a flexible substrate. That material, and three others, manganese bismuth, manganese antimony, and manganese aluminum germanium are candidates for plating on rigid substrates and may impact the lower end of the ultra-large storage market. It appears unlikely that the europium oxide media, though it requires less power for switching, can be produced which can operate practically at room temperature; the market is not likely to accept a system requiring cryogenic operating temperatures. These technologies have been widely discussed. For example, Eshenfelder[5] has prepared a useful

* Trademark

TABLE II

POTENTIAL MEDIA FOR MAGNETOOPTIC MEMORIES

MATERIAL	ADVANTAGES	DISADVANTAGES	CURRENT STATUS	POTENTIAL
Iron Silicide	(1) Curie Point ~110°C (2) Large Magnetooptic rotation(F)	(1) Anisotropy in Plane of Film (2) Room temperature coercivity (95 Oe)is low for high density storage	Exploratory	Further materials work required.
Europium Oxide	(1) Good 2F/α (figure of merit, α is absorption coefficient) (2) Can be prepared on variety of substitutes (3) Low Δ T (4) Low power requirement	Low operating temperature (77°K) due to low Curie Point	Feasibility of Technique demonstrated	Practicality yet to be demonstrated, further materials work required
Iron Doped Europium Oxide	(1) Similar to EuO magnetooptically	(1) Operating temperature (~180°K) low (2) Low coercivity	Exploratory	Practicality yet to be demonstrated, further materials work required
Europium Sulfide	(1) Stability (2) Ease of fabrication (3) Large M.O. rotation	Very low operating temperature (< 50°K)	Exploratory	Needs further work
* Manganese Bismuth	(1) Good 2F/α (2) High coercivity (3) Anisotropy perpendicular to plane (4) Demonstrated 10^8bit/in^2 density in material (5) Can be used in either Kerr or Faraday modes.	(1) Phase transition near Curie Point, 360°C (2) Fabrication complicated for mass production (3) Δ T = 350°C (4) Impractical for flexible media (5) Performance function of substrate (6) Requires high peak power (7) Efficiency about 0.01%.	Development of feasibility machine	Useful for high modulus substrate storage, card, disc, etc. Practicality yet to be demonstrated. Possible use in holographic mode, though 10KW instantaneous power required for 10^4bit page.
Manganese Arsenide	Low transition temperature (40°C)	(1) 2F/α lower than MnBi. (2) Recording properties unreported (3) Complicated preparation required.	Exploratory	Further materials work required.
*Manganese Antimony	(1) Good range of write temperature (2) Better media reliability than MnBi.	2F/α less than 50% of MnBi.	Very early. Exploratory. Films yet to be developed.	Further materials work required.
Gadolinium Iron Garnet	(1) Low ΔT (2) Low power requirement	(1) Single crystal material. (2) Relatively high media cost (3) Optical absorption < MnBi EuO.	Early exploratory	Practical media techniques yet to be developed. Could b^ put on discs.
* Cobalt- ** Phosphorous	(1) Simple plating process capable of mass production (2) Only material shown feasible for flexible substrate.	(1) Low M.O. rotation (2) Special readout technique required.	Write, read in engineering model demonstrated.	10^{13} bit mass memories on tape or 10^{10} -10^{11} bit disc memories. Possible use in holographic mode.
Manganese Aluminum Germanium	(1) Curie Point ~ 450°C (2) May be feasible for flexible substrate.	α ~ 0.1	Exploratory	Needs further work.

* Appears to be practical for discs.
** Appears to be practical for tape systems.

review. Treves,[14] Smith,[15] Stoffel and Schneider,[16] Fan and Grenier,[17] Aagard, et al.,[18] Wider et al.,[19] and Sherwood et al.[20] provide additional information.

a. *1966-1971. No magneto-optic systems available.*

b. *1976.* Since engineering feasibility of a magneto-optic system using cobalt-phosphorous on mylar, has been demonstrated, it seems evident that a system can be made available to the market if further development is funded. Performance, as indicated in Table I would be comparable or superior to that of the magnetic tape system available then. A significant advantage of magneto-optic systems is that it appears to be practical to achieve packing densities about ten times higher than is now practical with magnetic tape systems. Secondly, there is no contact between the write/read transducer and the medium. Major disadvantages are the handling of the media, the reliable positioning of the write and read photon (laser) beams, and availability of reliable lasers of the required power at appropriate costs. The demands placed upon mechanical positioning controls to take advantage of the higher packing density potential are severe particularly with respect to long failure-free life. There is a fundamental limit to storage density imposed by the diffraction limit of the wavelength of light used. If all elements were perfect, including hundreds of square feet of defect-free storage surface, and $f/1$ lenses used, one bit per wavelength stored with no gap between bits would achieve about $2(10^9)$ bits per square inch. Practical considerations, exclusive of the errors imposed by the storage surface, limit achievable storage densities to about $3(10^7)$ bits per square inch. *This practical limit applies to all of the light actuated memory systems.* Electron beam systems, yet to be described can ideally achieve higher packing densities because the wavelength of the electron beam is about 1/5000 that of light beams. Here, too, practical considerations limit packing density to about the same level.

c. *1981.* If magneto-optic systems are developed for 1976 availability, and they prove to be practical in use, then enough improvements are possible that, if funded, a system having the performance indicated in Table I is possible by 1981.

3. *Photographic, Silver Halide Tape, Using Electron or Laser Beam Write and Read, Permanent*

The technology required for electron and photon beam write/read systems for recording on silver halide media has been developed. Kuehler and Kerby[21] reported on the IBM 1360, flying spot scanner read system. Bousky and Diermann[2] and Pass and Wood[22] reported on electron and laser beam analog recorders for instrumentation purposes. Generally, electron beam systems permit the highest data rate. Although, electron beams can record at the highest packing density, engineering limitations associated with accurately positioning the readout beam onto a flexible medium dictate much lower packing densities; they become comparable to those of magnetic recording systems.

Electron beam systems require operating in vacuum, and laser beam systems require complex light deflection components. Nevertheless, there has been substantial user experience with both kinds of systems for storing analog information.

a. *1966.* No electron or laser beam systems were available.

b. *1971.* In 1965 the IBM 1360[23] was contracted to be built. From 1967 to date five units have been delivered. Penny et al.[24] described the use of such a system, and Burnham[25] the medium. IBM produced systems under contract, but is not offering them for sale, presumably because of profit (production cost, market, reliability) considerations.

Times are different for write, which is by an electron beam in a vacuum, and read, which is by a flying spot scanner. Wet development is automatic. Data is stored on chips which are accessed in pneumatic tubes and stored in cells, which in turn, are stored in trays. Costs shown are estimated.

c. *1976, 1981.* Data given in Table I for these years reflect the performance characteristics achievable by laser systems if their development is funded. Since the media is not erasable, and off-line processing is required, *it seems unlikely that such digital systems will be produced.* Analog systems, however, will probably continue to be produced.

4. *Metallized Tape, Electron or Laser Beam Hole-Burning Write, and Readout, Permanent.*

Dove described a hole-burning storage system in 1964, obtained a patent in 1965, and described the technology recently.[4] He predicted physical feasibility of 10^9 bits per square inch storage capacity, and identified the problems of (1) plated media cracking due to differential coefficients of expansion of the plating and substrate, and (2) preferential expansion due to fabrication. Engineering considerations limit packing density to not much more than 10^7 bits per square inch.

a. *1966.* No system available.

b. *1971.* Precision Instruments announced the Unicon,* a metallized tape hole-burning system in 1969. Dell[26] most recently described the laser beam system which accesses strips of tape from a file and wraps them around either of two rotating drums. The system, though data is not erasable, does permit adding data in spaces not previously written upon. It does not permit graceful degradation. Customer acceptance tests are scheduled for November, 1971. Although the recording material is much more expensive than magnetic tape, the higher packing density yields per bit costs comparable to conventional tapes. Microinch level position accuracy, mechanical part upon mechanical part, is required to achieve the stated error rate. Table I lists the announced performance.

c. *1976-1981.* If Unicon* systems are delivered, have sufficient reliability, and are able to

* Trademark

* Trademark

achieve satisfactory error rates (there are drop-in problems in which dust, or imperfections lead the readout system to read ones rather than zeros), improvements are likely to be made. Since the announced system contains packing density near to the practical optical limit, packing density is unlikely to be improved, and, in fact, may have to be reduced in order to improve error rate. Improvements are most likely to occur in data rates, error rates, and in larger file storage. Performances that seem possible are shown in Table I.

B. *Holographic Systems*

1. *Photographic, Silver Halide Tape, Laser Beam Write and Read, Permanent.*

Anderson[27] presented the most comprehensive review of holographic digital storage, erasable and permanent, on surfaces and in volumes. He observed that, though densities exceeding 10^6 bits per square inch can be written, and readout occurs without critical alignment and tight mechanical tolerances, in situ read and write, but not erase is presently practical. It appears to be practical to develop 10^8 bit data blocks with throughput exceeding $500(10^6)$ bits per second and random access less than 2 microseconds to photo diode readout arrays. In the distant future, it may be possible to write and read out 10^{11} bits in a volume, though selective erasure may not be attainable. Readers are referred also to Briggs,[28] Chen and Tufte,[3] Eshenfelder,[5] and Rajchman.[29]

Problems remaining to be solved relate to developing capability for maintaining mechanical and dimensional stability on a flexible medium or adapting optical or electronic circuits to compensate for imperfections, and inventing and developing suitable light valves, page composers, lasers, deflectors, and photodetectors. Because of the first of these, it appears likely that holographic storage will first be achieved on plates. If developed, it is likely the dimensional stability and alignment problems will probably dictate somewhat lower packing density than is achievable on rigid surfaces. Estimates of what is achievable in a system most likely to become available in 1977/1978, if development is funded, are shown in Table I. If the problems can be solved so that performance comparable to storage on rigid surfaces is achievable, then systems with 25 times the storage capacity and correspondingly larger sizes and lower costs will be achieved. The highest packing density of any system should be achievable, and this would then be the most practical nonerasable store.

2. *Plated Tape, Magneto-optic, Erasable.*

If present applied research efforts continue, it is possible that a satisfactory medium which permits in situ write, read, and erase may be developed within 5 to 10 years. If so, if problems listed above are solved, and if funded, a system with performance described in Table I might be ready by 1981. Again, first applications are more likely to be large memories on rigid surfaces. If the dimensional stability problems are solved to permit storage comparable to that on rigid surfaces, a storage system such as this would probably be the most satisfactory of all proposed. However, there are too many ifs to believe this will happen much before 1981; it may not occur until a few years later.

CONCLUSIONS

Ultra-large digital storage systems will probably continue to be predominantly magnetic tape, inductor-head write/read systems through the next ten years. System improvements are practical and should serve to make it impractical for competing technologies to be developed to the point where they can provide significant competition. For example, electron beam systems with extremely high packing densities are likely to be impractical and not developed because of beam control problems. Holographic storage systems, while practical for fiche-like permanent storage systems, are limited by dimensional and positioning problems, and by the likelihood that an appropriate erasable medium may not be developed soon enough within this time period for such a system to become available. The erasable holographic storage system appears to have the greatest potential for the second 10 year period.

Two ultra-large storage systems have been produced, one erasable (TBM*) the other a permanent store (Unicon*). Time will tell the practicality of these systems.

REFERENCES

1 J HARKER
Large storage systems using rigid media, past, present and future
SJCC 1972

* Trademark

2 S BOUSKY J DIERMANN
Characteristics of scanning laser and electron beams in bulk data storage
IEEE Intermag Conference 1971 Paper 19.2

3 D CHEN O N TUFTE
Optical memories, now and in the future
Electronics World V 84 No 4 pp 56–60 Oct 1970

4 J F DOVE
Electron and laser beam storage and retrieval
AGARD Opto-Electronics Signal Processing Techniques pp 10–1 to 10–7 Feb 1970

5 A H ESCHENFELDER
Promise of magneto-optic storage systems compared to conventional magnetic technology
J App Phys V 41 pp 1372–1383 1970

6 J W WEIL
An introduction to massive stores
Honeywell Computer Journal V 5 pp 88–92 1971

7 R P HUNT
A magnetoresistive readout transudcer
IEEE Trans Magnetics V Mag 6 pp 602–603 1970

8 A S HOAGLAND
Mass storage revisited
AFIPS FJCC Proceedings V 31 pp 255–260 1967

9 D T BEST
The present and future of moving media memories
IEEE International Convention Digest pp 270–271 1971

10 V E RAGOSINE J C MALLINSON
Bulk storage technology magnetic recording
IEEE Intermag Conference Paper 19.3 1971

11 A F SHUGART Y H TONG
IBM 2321 data cell drive
SJCC Proceedings V 28 pp 335–345 1966 See also
IBM System/360 component descriptions—2841 and associated DASD
IBM Systems Reference Library San Jose California

12 R B GENTILE J R LUCAS JR
The TABLON mass storage network
AFIPS SJCC pp 345–356 1971

13 Editor
Datamation p 18 Sept 1 1971

14 D TREVES
Magneto-optic detection of high density recording
J Appl Phys V 38 pp 1192–1196 March 1967

15 D O SMITH
Proposal for a magnetic film memory accessed by combined photon and electron beams
IEEE Trans on Magnetics V Mag 3 pp 593–599 1967

16 A M STOFFEL J SCHNEIDER
Magneto-optic properties of Mn-As films
J Appl Phys V 41 pp 1405–1407 1970

17 G FAN J H GRENIER
Low temperature storage using Ga as lasers and EuO as storage medium
J Appl Phys V 41 pp 1401–1403 1970

18 R L AAGARD D CHEN et al
Optical mass memory experiments on thin films of Mn Bi
IEEE Trans Magnetics V Mag 4 pp 412–416 1968

19 H WIEDER S S LAVENBERG G J FAN R A BURN
A study of thermomagnetic remanence writing on EuO
J Appl Phys V 42 pp 3458–3462 1971

20 R C SHERWOOD et al
Mn Al Ge films for magneto-optic application
J Appl Physics Vol 42 pp 1704–1705 1971

21 J D KUEHLER H R KERBY
A photodigital mass storage system
AFIPS FJCC Proceedings V 29 pp 735–742 1966

22 H W PASS J WOOD
100 MHz and 100 MBIT recording and readout instrumentation systems for electromagnetic field perturbations
IEEE Trans on Nuclear Science V NS–18 pp 118–123 1971

23 R M FURMAN
IBM 1360 photo-digital storage system
TR02.427 IBM SDD San Jose California May 15 1968

24 S J PENNY R FINK M ALSTON-GARNJST
Design of a very large storage system
AFIPS FJCC Proceedings pp 45–51 1970

25 D C BURNHAM
Microfilm as computer memory
Computer Handling of Graphic Information Proceedings pp 137–142 1970

26 H R DELL
Design of a high density optical mass memory system
Computer Design V 10 No 8 pp 49–53 Aug 1971

27 L K ANDERSON
Application of holographic optical techniques to bulk memory
IEEE Intermag Conference Paper 19.4 1971

28 D A BRIGGS P WATERWORTH
Holographic data recording systems
Electro-Optics International Conf Proceedings pp 179–189 1971

29 J A RAJCHMAN
Promise of optical memories
J Appl Phys V 41 pp 1376–1383 1970

New devices for sequential access memory

by F. H. BLECHER

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

ABSTRACT

Two classes of devices are seriously challenging the electromechanical disk file for bulk storage applications—magnetic bubble and charge-transfer. These devices can be used for sequential access memory and feature solid-state reliability, small size and potential low costs.

Magnetic bubble devices consist of a thin layer of uniaxial magnetic material imbedded in a bias magnetic field. The easy direction of magnetization is perpendicular to the surface of the thin layer and is in the same direction as the bias field. There is a range of bias fields for which regions of magnetization exist in the uniaxial material with a magnetic polarity opposite to that of the bias field. These regions of reverse magnetization have the form of right circular cylinders and are known as magnetic bubble domains or simply bubbles. The bubble is a stable entity that can be propagated by use of current carrying conductors on the surface of the thin magnetic layer or by the application of an in-plane rotating magnetic field which is used to magnetize small permalloy features on the surface of the magnetic layer (field access).

Rare earth garnet magnetic materials are now available in very thin layers (2 to 5 μm) with the necessary anisotropy and magnetization to produce extremely small bubble domains (4 to 10 μm diameter), and with very few defects to impede domain motion ($\sim$2 cm^{-2}). These materials should make possible storage densities of the order of several million bits per square inch. In this talk, the design of bubble circuits using single conductor current propagation is discussed with particular application to a multi-megabit bubble file.

Two semiconductor devices are available which serve as dynamic-shift registers. One is of a functional nature and is called charge-coupled device (CCD). This device operates on the principle of charge transfer along the interface between a semiconductor and an insulating layer. The charges are moved by appropriately controlling the potential of metal electrodes placed on the surface of the insulator. The other semiconductor device is an integrated version of the IGFET bucket-brigade shift register. Recent developments in both of these technologies are discussed with application to large sequential access memories.

Two direct methods for reconstructing pictures from their projections—A comparative study

by GABOR T. HERMAN

State University of New York
Buffalo, New York

INTRODUCTION

There are situations in the natural sciences and medicine (e.g., in electron microscopy or X-ray photography) in which it is desirable to estimate the gray levels of a picture at individual points from the sums of the gray levels along straight lines (projections) at a few angles. Usually, in such situations, the picture is far from determined and the problem is to find the "most representative" picture of the class of pictures which have the given projections.

In recent years, there has been a very active interest in this problem. A number of algorithms have been proposed for solving it. The algorithms are applicable in a large and varied number of fields. This is perhaps best demonstrated by the reference list, where we see that such algorithms have been published or discussed in journals on computer science, theoretical biology, molecular biology, cellular biology, biophysics, medical and biological engineering, roentgenology, radiology, chemistry, optics, crystallography, physics, electrical engineering, as well as in general science journals. The most important use of the algorithms is the reconstruction of objects (e.g., biological macromolecules, viruses, protein crystals or parts of the human body) from electronmicrographs or X-ray photographs.

Most of the effort in this area has so far been concentrated in developing algorithms and in applying these algorithms for actual reconstructions. So far, there has been only limited attention paid to the relative merits of the algorithms. The present paper is a report on the first of a series of comparative studies of various algorithms for reconstructing pictures from their projections. The two techniques that we shall compare are one of the algebraic reconstruction techniques (ART) of Gordon, Bender & Herman (1970) and a summation technique, which has been independently discovered by a number of people, but the most detailed previous study of which has been given by Vainshtein (1971a, b).

Both these techniques are direct techniques in the sense that the reconstruction is being done entirely in real space (density space), without the use of Fourier transforms. This is worth pointing out, since the first applicable method for reconstructing pictures from their projections (discovered independently by DeRosier & Klug [1968] and Tretiak, Eden & Simon [1969]) depended on the fact that the Fourier transform of a projection is a section of the Fourier transform of the picture. Thus, reconstruction was achieved by taking Fourier transforms of the projections, interpolating in Fourier space to get an approximation of the Fourier transform of the picture, and then taking an inverse Fourier transform to get an approximation of the picture itself. Detailed descriptions of such techniques were given by Crowther, DeRosier & Klug (1970) and Ramachandran (1971).

Crowther, DeRosier & Klug (1970) state that "for general particles and methods of data collection, the method of density space reconstruction is computationally impracticable on presently available computers." However, they appear to be wrong on this point. Both the methods discussed in this paper have been implemented, and precise timings will be given on a number of examples to show that they are not computationally impracticable. Even more interestingly, expanding the mathematics on which the Fourier techniques are based, Ramachandran & Lakshminarayanan (1971a, b) succeeded to produce another direct method, called the convolution method, for the reconstruction of pictures from their projections. They demonstrate on a number of idealized pictures that their technique is faster than the Fourier techniques and that it also gives a better reconstruction. Detailed comparative studies between ART, the convolution method and the Fourier techniques are presently under way, and will be reported on in later publications.

In the next section, we shall briefly describe the problem of reconstruction of objects from their projections and discuss the criteria we shall use for evaluating algorithms for solving this problem. This is the first exhaustive discussion of such criteria, it will form the basis of later comparative studies as well. The following two sections will give brief descriptions of ART and the summation method, respectively. In particular, we shall give a version of the summation method which is superior to previously published versions. In the final section, we shall give a report on the results of our experiments.

A basic difference between the two techniques is that the summation method takes a fixed length of time to run, while ART is an iterative technique such that its result can be improved by repeated iterations. One iteration of ART costs about the same as the summation technique. In rough terms, the results of our experiments indicate the following.

If the number of projections is small, one iteration of ART seems to perform somewhat worse than the summation method. As the number of projections increases, the relative performance of one iteration of ART as opposed to the summation method improves, and eventually it produces superior results. The break even point seems to be at eight projections. However, by repeated iterations, the performance of ART can be improved and it eventually surpasses the summation method in all our experiments. The improvement in ART is especially impressive with eight or more projections. In such cases, we obtain, with five iterations of ART, results which are far superior to the results of the summation method.

METHOD OF COMPARISON

First we wish to describe briefly the problem of reconstructing pictures from their projections. A more detailed description of the problem can be found, for example, in Frieder & Herman (1971).

We assume that f is a function of two real variables such that $f(x, y) = 0$ outside a square region in the first quadrant of a rectangular coordinate system (see Figure 1). Further, we assume $0 \leq f(x, y) \leq 1$, everywhere. Following Rosenfeld (1969), we shall call a function with such properties a *picture function*. ($f(x, y) = 0$ means white, $f(x, y) = 1$ means black, with other *gray levels* in between.)

Let θ be an angle, $-90° < \theta \leq 90°$, and suppose that l is a line making an angle θ with the positive x axis. Suppose, further, that l is divided into equal line segments and that we draw lines perpendicular to l from the points separating these segments. Thus, we get a number of infinite bands, partitioning the (x, y)-plane. We shall refer to these bands as the *rays* of the *projection* associated with θ. Suppose there are r_θ rays which actually intersect the rectangle. For $1 \leq k \leq r_\theta$, let $R_{f,k,\theta}$ denote the integral of $f(x, y)$ in the k'th band which intersects the rectangle. (The integral need only be carried out in the shaded region in Figure 1, since $f(x, y) = 0$ elsewhere.) We shall refer to $R_{f,1,\theta}, \ldots, R_{f,r_\theta,\theta}$ as the *ray-sums* of the projection associated with θ.

The problem of reconstructing pictures from their projections can now be stated as follows.

Give an algorithm which, given

(1) the position of the square region within which $f(x, y) \neq 0$;
(2) angles $\theta_1, \theta_2, \ldots, \theta_m$, together with the widths and position of rays of the projections associated with these angles;
(3) the ray-sums of the projections associated with $\theta_1, \theta_2, \ldots, \theta_m$;

will produce a function f' which is a good approximation of f.

One can only hope for an approximation, since it has been proved, even for the limiting case when the ray

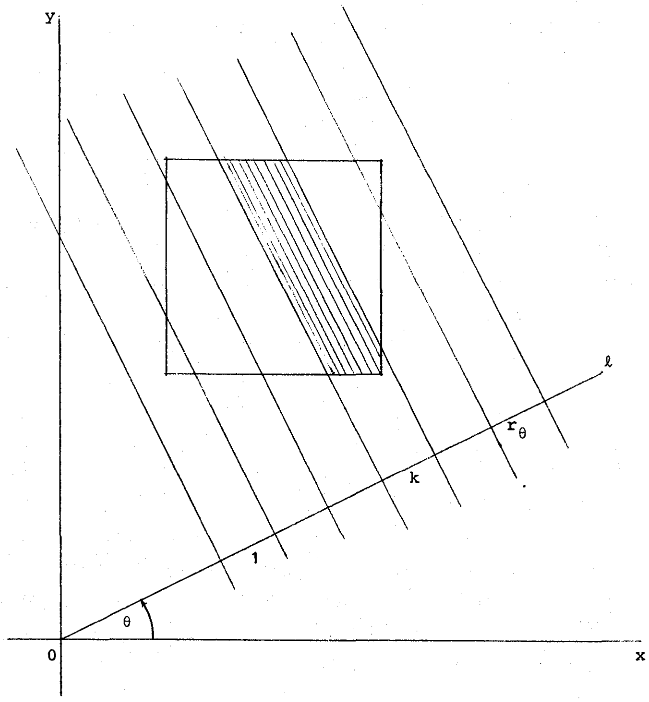


Figure 1

widths tend to zero, that a non-trivial picture cannot be uniquely determined from its projections (Frieder & Herman [1971], Theorem 1). The algorithm ought to be such that for pictures that we are likely to be interested in, they will provide a good enough approximation. This idea is discussed in some detail by Frieder & Herman (1971). For the purpose of the present paper, the following should suffice.

As will be seen in the next two sections, reconstruction algorithms tend to be somewhat heuristic. Hence, an analytical estimate of their performance is somewhat difficult to obtain. Even if there was a method to do it, we would be faced with the problem that the functions f, to which the algorithms may be applied in practice, form a small but not clearly defined subset of the set of all picture functions f. Hence, analytical methods, which give equal weight to all picture functions, may give much worse estimates than what one would obtain in practice. Therefore, it appears reasonable to evaluate and compare reconstruction techniques on the basis of their performance on selected typical pictures and test patterns. This is, indeed, what we are going to do in this paper.

Since our algorithms are to be implemented on a digital computer, it is reasonable to demand that the output f' be a digitized quantized picture function (Rosenfeld [1969], Section 1.1). Indeed, in all our examples, f' will be defined on a 64×64 matrix, and can assume as many values between 0 and 1 as the accuracy of the computer allows. In pictorial representation, we shall use only 16 equally spaced gray levels between 0 and 1, inclusive.

For convenience, we shall use as test data only picture functions which are of the same kind, i.e., 64×64 digitized pictures with 16 equally spaced quantized gray levels. When working out the ray-sums, we shall sum the values of $f(x, y)$ at those points which lie within the ray. At first sight, it may appear that this may change the problem in an essential way, but we shall show in the Appendix that this is not so.

In all our experiments, the rays are defined in such a way, that it is true for at least one edge of the 64×64 matrix that each of the points on that edge is a midpoint of a ray. The reasons for this are discussed in Frieder & Herman (1971) and, more briefly, in the Appendix.

We have decided to use four test patterns (Plate A). The first two of these are binary valued picture functions (black and white) which have been used by Vainshtein in his demonstrations of the summation method. (A1 comes from Vainshtein [1971a] and A2 from Vainshtein [1971b].) We are not aware of any other previously published test patterns on which the summation method was demonstrated. The situation is better concerning ART, Gordon, Bender & Herman (1970), Herman & Rowland (1971) and Bellman, Bender, Gordon & Rowe (1971), all contain a number of test patterns showing the operation of ART. We have decided to use two of these for the present comparative study, both half-tone (i.e., all 16 gray levels occur in them). One is the face of a little girl, Judy (A3), the other is the photograph of some stomata (A4). The first of these has been used by Gordon, Bender & Herman (1970) and Gordon & Herman (1971), the second has been used by Herman & Rowland (1971). Since in previous studies these pictures were not digitized at 64×64 points, we kept the previous digitization (49×49 and 50×50 points) and inserted them into a white frame. This makes comparison with previously published tests concerning these patterns easier.

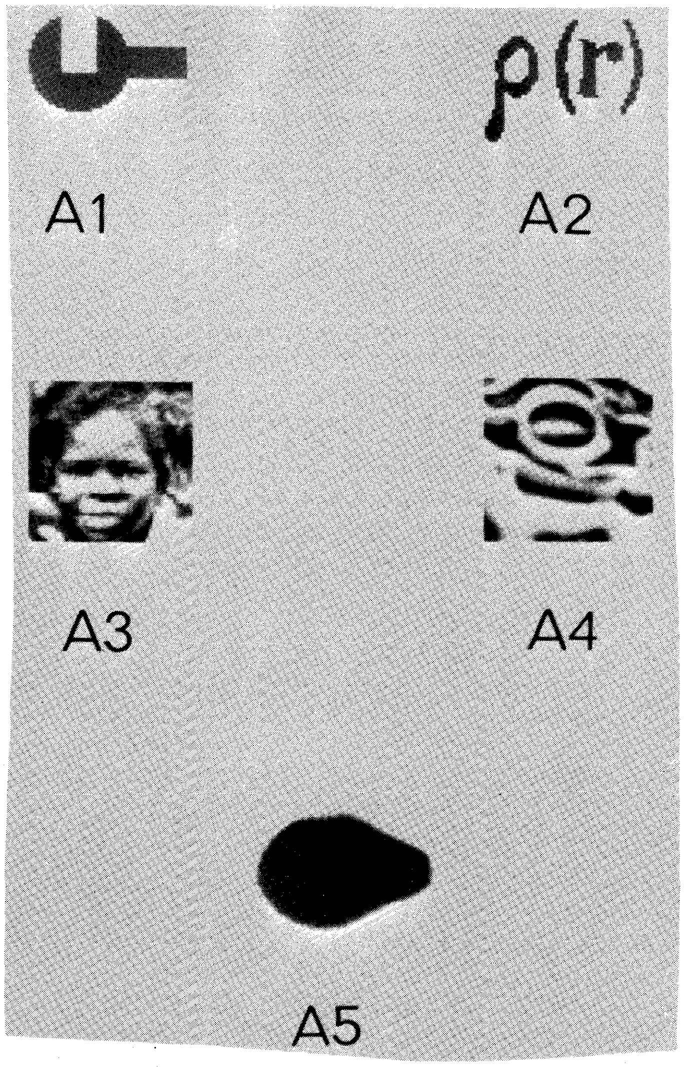


Plate A

For each of the four pictures, we have carried out reconstruction operations using four different sets of projection angles. The first three sets have been used by Vainshtein in his demonstration of the summation method. In all three sets the projection angles are equally spaced between −90° and +90° (with 90° being one of the angles), and there are 4, 8 and 30 angles, respectively. We used these sets of angles, so that our results can be directly compared with the results of Vainshtein (1971a, b).

The fourth set of projection angles comes from a small range. This is particularly important in electronmicroscopic applications, because of the restricted range of the tilting stage. We have, in fact, used the same set of projection angles that have been used by Bender, Bellman & Gordon (1970) in their reconstruction (using ART) of ribosomes from electronmicrographs. We have rotated these angles so that none of them is near 0° or 90°. The reason for this is that both ART and the summation technique would tend to give unusually good results if the edges in a test pattern (like A1) are parallel to one of the projection angles. The fourth set of projection angles which we used was actually 35°, 44°, 62°, 80°, 98° (=−82°) and 116° (=−64°). Thus, all these views come from a range of 81° as opposed to a full range of 180°.

Four pictures, with four sets of projections each, provide us with 16 reconstructions to be carried out with each of the techniques we are investigating. Then we are faced with the problem of evaluating the success of the reconstructions. There is no standard way of doing this. We have decided to use four different sets of criteria: overall nearness of the original and reconstructed pictures, resolution of fine detail, visual evaluation, and cost of reconstruction. We shall now discuss these criteria in some detail.

Overall nearness

(1) Gordon, Bender & Herman (1970) and Gordon & Herman (1971) used the root mean square distance δ as a measure of the overall nearness of the original and reconstructed pictures.

$$\delta=\left[\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(f(x_{i,j},y_{i,j})-f'(x_{i,j},y_{i,j})\right)^2\right]^{1/2},$$

where $x_{i,j}$ and $y_{i,j}$ are the x and y coordinates of the (i,j)'th point in the $n\times n$ matrix. The validity of such a measure has been questioned by Crowther & Klug (1971), but the discussion by Frieder & Herman (1971, especially Theorem 2) shows that δ is a very reasonable measure for the overall performance of a reconstruction technique.

(2) Ramachandran & Lakshminarayanan (1971a, b) use another measure R of overall nearness in their comparison of the Fourier techniques with their convolution method. This is adopted from X-ray crystallography, and we shall work it out as well as the δ measure for all our experiments.

$$R=\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left|f(x_{i,j},y_{i,j})-f'(x_{i,j},y_{i,j})\right|}{\sum_{i=1}^{n}\sum_{j=1}^{n}f(x_{i,j},y_{i,j})}.$$

Thus, R is the mean relative error of the reconstruction.

(3) For a number of applications one wants to make the assumption that f' is a binary valued picture function. This is obviously so if we know that f is binary valued. Even if f is not binary valued, we may wish to contour f' to get some idea of the overall shape of the object which is in the picture f. Let

$$\bar{f}(x,y)=\begin{cases}1, & \text{if } f(x,y)\geq 0.5,\\ 0, & \text{if } f(x,y)<0.5,\end{cases}$$

and let $\bar{f}'$ be similarly defined. We shall refer to $\bar{f}$ and $\bar{f}'$ as the contoured versions of f and f'. (Naturally, for a binary valued picture function f, $f=\bar{f}$.) We shall also work out the mean relative error, $\bar{R}$, for the contoured versions of f and f'.

$$\bar{R}=\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left|\bar{f}(x_{i,j},y_{i,j})-\bar{f}'(x_{i,j},y_{i,j})\right|}{\sum_{i=1}^{n}\sum_{j=1}^{n}\bar{f}(x_{i,j},y_{i,j})}.$$

The measure $\bar{R}$ has also been used by Chang & Shelton (1971) in comparing two algorithms for binary pattern reconstructions.

(4) A property of the output of a reconstruction should be that it is consistent with the information available to the algorithm. In other words, the ray-sums of the reconstructed picture should be the same as the ray-sums of the original. Even though there are algorithms which achieve this (Gordon & Herman [1971], and, for binary valued pictures only, Chang [1970], Chang & Shelton [1971]), they usually tend to be slow for practical applications. Both ART and the summation technique will produce pictures where the ray-sums will only approximate the given data. One appropriate measure of the overall nearness of the reconstruction is the root mean square distance ρ

between the ray-sums on which the reconstruction is based and the ray-sums of the reconstructed picture.

$$\rho=\left[\frac{1}{\sum_{l=1}^{m} r_{\theta_l}}\sum_{l=1}^{m}\sum_{k=1}^{r_{\theta_l}}(R_{f,k,\theta_l}-R_{f',k,\theta_l})^2\right]^{1/2},$$

where m is the number of projections and R_{f',k,θ_l} denotes the ray-sum in the k'th ray of the l'th projection for the reconstructed picture f'.

(5) A simple way of deciding the overall success of a reconstruction is to ask whether the reconstructed pattern resolves the original. Based on a suggestion of Harris (1964), Frieder & Herman (1971) used the following criterion for answering this question. A reconstruction resolves a picture, if the output is nearer (in the sense of δ) to the original than to uniform gray. Since the mean gray level of input and output is the same both for ART and for the summation technique, the distance from uniform gray of the output is nothing but the standard deviation of the output. In our experiments, we shall say that the output of the reconstruction resolves the original if, and only if, the standard deviation of the output is greater than δ.

Resolution in fine detail

The major objection of Crowther & Klug (1971) to δ as a measure of difference between the reconstruction and the original is that "it reaches a low value once the large scale features are correct and is relatively insensitive to errors in the fine details." As it was pointed out by Frieder & Herman (1971), this objection is not quite valid. However, it is reasonable to ask what is the guaranteed maximum error in the average grayness of a region of certain size in the reconstructed picture. With this idea in mind, Frieder & Herman (1971) introduced the notion of (a, l, ϵ)-resolution. They said that two picture functions f and g with $4a^2$ non-zero area (a, l, ϵ)-resolve each other if, for every square region of size l^2, the mean grayness of the pictures in that region differs by less than ϵ.

The problem with (a, l, ϵ)-resolution is that it is difficult to calculate. For example, if we wanted to find the minimal ϵ such that an original and a reconstructed 64×64 picture $(32, 8, \epsilon)$-resolve each other, we would have to work out the mean gray value of nearly 6000 regions, each with 64 points. We have therefore devised another method, which is a good approximation to (a, l, ϵ)-resolution, but which is computationally much simpler.

For $0\leq l\leq 6$, let

$$\epsilon(l)=\max_{\substack{0\leq i\leq(64/2^l)-1\\ 0\leq j\leq(64/2^l)-1}}\left[\frac{1}{2^{2l}}\left|\sum_{1\leq m\leq 2^l,1\leq n\leq 2^l}\left[f(x_{i\cdot 2^l+m}, y_{i\cdot 2^l+n})-f'(x_{i\cdot 2^l+m}, y_{i\cdot 2^l+n})\right]\right|\right]$$

In other words, $\epsilon(l)$ is the maximum difference between f and f', when they are both digitized as $(64/2^l)\times(64/2^l)$ pictures. Thus, $\epsilon(0)$ is the maximum difference between the values at any point of f and f', while $\epsilon(6)$ is the difference between the mean gray values of f and f'. For every l, $\epsilon(l)$ indicates the point by point reliability of the reconstruction when both the original and the reconstructed picture are digitized as $(64/2^l)\times(64/2^l)$ pictures. For example, if $\epsilon(2)=.01$, and the gray level in the 16×16 digitized version of the reconstructed picture is .253 at a certain point, then we know that the gray level in the 16×16 digitized version of the original is between .243 and .263.

The notion expressed above is easily generalized for test patterns which are digitized as $2^k\times 2^k$ matrices, for any k. Also, $\epsilon(l)$ is computationally easy to calculate, it only requires a repeated increase of the roughness of digitization by a factor of two.

For each of our experiments, we shall give the values of $\epsilon(0), \epsilon(1), \ldots, \epsilon(5)$. $\epsilon(6)$ is always zero.

An alternative way of representing our results regarding resolution in fine detail is to state the size of detail which is reliable within a certain error. For any $\epsilon>0$, $l(\epsilon)$ will denote the maximum number n, such that n is a power of two and the $n\times n$ digitization of the original and the reconstructed pictures differ by less than ϵ at all points. Thus, $l(0.1)=32$ will mean that the original and the reconstructed pictures digitized at 32×32 points will differ at each point by less than 0.1, but if they are digitized at 64×64 points, then they differ by more than 0.1 at least at one point. So, $l(\epsilon)$ is roughly the resolution of the reconstruction if the error tolerance is ϵ. For each of our experiments, we shall work out $l(0.01)$, $l(0.02)$, $l(0.05)$, $l(0.10)$, $l(0.25)$ and $l(0.50)$. Clearly, there is no point in calculating $l(\epsilon)$ for $\epsilon>0.5$.

Visual evaluation

The methods mentioned in (a) and (b) above include all computationally possible ways of measuring the success of reconstructions which are known to us. We left out measuring resolution by the use of Fourier transforms (see (a, l, ϵ)-resolution in Fourier space in the paper by Frieder & Herman (1971)), because of its computational difficulty, and its inappropriateness for comparing two direct methods. Even though the mathematical measures we used should provide quite conclusive evidence of the relative merits of two reconstruction techniques, there has not yet been enough experience with them to know the correlation between the values of the measures and visual acceptability of the reconstruction. For this reason, we shall give for each of our experiments the reconstructed picture, which can then be compared with the original. When the test-pattern is a binary valued picture function, we shall also give the contoured version of the reconstructed picture.

Cost of reconstruction

This is obviously an important measure. In practical applications, especially three-dimensional reconstruction, one has to carry out a large number of reconstructions, and cost can be a prohibitive factor.

To insure that our cost comparisons are valid, we have incorporated both ART and the summation method in the same general program written in FORTRAN. They make use of the same set of service subroutines, e.g., the one for finding out which points lie in a particular ray.

Time, rather than cost is given, since it is easier to obtain. However, cost appears to be a more stable measure between installations. Since all experiments have been run on a CDC 6400, at a cost of $475 per hour, the reader can easily work out the actual cost of the runs. (Rate may vary from installation to installation.) Run time is given in seconds.

DESCRIPTION OF ART

The method we shall describe now is one of the algebraic reconstruction techniques of Gordon, Bender & Herman (1970), the one which was called in that paper the direct additive method. Other papers relevant to this method are Bender, Bellman & Gordon (1970), Crowther & Klug (1971), Bellman, Bender, Gordon & Rowe (1971), Frieder & Herman (1971) and Herman & Rowland (1971).

The basic idea of the method is the following. Starting from a blank picture, the ray-sums of all the projections are satisfied one after the other by distributing the difference between the desired ray-sum and the actual ray-sum equally amongst all the points in the ray. While satisfying the ray-sums of a particular projection, the process usually disturbs the ray-sums of previously satisfied projections. However, as we repeatedly go through satisfying all the projections, the disturbances get smaller and smaller, and eventually the method converges to a picture which satisfies all the projections. Because we start with a uniform blank, and while satisfying each of the projections we make as uniform changes as possible, the final product tends to be as smooth as a picture satisfying the given projections can possibly be. For practical applications, this seems to be a desirable property of a reconstruction algorithm. Roughly speaking, our reconstructed picture will show only features which are forced upon it by the projections, rather than features which are introduced by the reconstruction process.

Mathematically, let f' be the partially reconstructed picture just before we wish to satisfy the projection associated with the angle θ. Let $(x_{i,j}, y_{i,j})$ lie on the k'th ray of that projection, and let $N_{k,\theta}$ be the number of points on that ray. Then f' is changed into f'' by the rule

$$f''(x_{i,j}, y_{i,j}) = f'(x_{i,j}, y_{i,j}) + (R_{f,k,\theta} - R_{f',k,\theta})/N_{k,\theta}.$$

If the value of f'' obtained in this way is negative, it is rounded to zero, if its value is greater than one, it is rounded to one.

The process of satisfying all projections one after the other exactly once is referred to as one *iteration*. The accuracy of ART increases with the number of iterations, and we report on the results of experiments after 1, 5 and 25 iterations.

DESCRIPTION OF THE SUMMATION METHOD

This method has been described by Vainshtein (1971a, b), Gordon, Bender & Herman (1970), Gaarder & Herman (1971), Ramachandran & Lakshminarayanan (1971b). The most detailed discussion of its properties is contained in the work of Vainshtein (1971a, b).

Roughly speaking, the idea is to distribute each of the ray-sums equally amongst all the points in the corresponding ray. If there are m projections, this will result in a picture whose total density is m times what it should be. We therefore subtract from the value at each point $(m-1)d$, where d is the mean density of the picture. (The mean density can be worked out from the ray-sums.) Rounding negative values to zero and values greater than one to one, we get our first approximation to the input.

This procedure has a lot to recommend it. It is conceptually and computationally simple. It can be implemented by a photo-summation device without the use of a digital computer. It seems to provide us with a smooth picture. In fact, if the projections and rays satisfy certain conditions, the result of the algorithm will be a picture which satisfies the projections and for which the variance of gray levels is smaller than for any other picture satisfying the projections (see Gaarder & Herman [1971]).

However, the algorithm has some rather peculiar properties which make it often useless in practice. It appears that as a result of rounding to zero and one, the average density of the picture increases. Hence, we get, with 30 projections, the picture in A5 corresponding to the test pattern in A1. Vainshtein (1971a) recommends that the output should be contoured at a level so that the total density of input and output are the same. However, this advice can only be followed for binary-valued picture functions. To improve the quality of the output for arbitrary picture functions, we carried out the following process.

First of all, if a ray-sum is zero, we know that the value of the function at all points on that ray is zero. The first step in our modified algorithm is to mark all points which lie on rays with ray-sums equal to zero.

Then we carry out the process described above, but equally distributing the ray-sums only amongst the unmarked points in the ray. After subtracting $(m-1)d$ from the value at each point, we round all negative values to zero.

At this stage, the mean density $\bar{d}$ of the reconstructed picture will usually be greater than d. So we multiply the value of the reconstructed function at each point by $d/\bar{d}$, making the mean density of the reconstruction equal to that of the original. We should at this point round all values greater than 1 to 1, but there was no need for this in any of the 16 experiments that we tried.

We found that this modified summation method gave much better results than the simpler technique described at the beginning of the section. For example, for the test-pattern A1 and 30 projections, we get the following comparisons.

	δ	$\epsilon(0)$	$\epsilon(2)$	$\epsilon(4)$	time
simple method	0.32	1.0	1.0	0.44	11.25
modified method	0.18	0.75	0.57	0.09	12.92

Clearly, the large improvement is well worth the small additional cost. In all the experiments reported on in the next section, we used the modified summation method.

A final comment. ART and the summation method have been referred to (Crowther & Klug [1971]) as "very similar." The descriptions above clearly indicate that this is not so, and, as can be seen from the next section, the power of the two methods is quite different. It is therefore invalid to draw conclusions about the behavior of one of these methods from the observed behavior of the other one. The two methods could be combined in an iterative summation method, where after each iteration the difference between the desired ray-sums and actual ray-sums is simultaneously equally distributed amongst the points on the rays. This method should give similar results to ART, but it requires a considerably larger memory in a digital computer implementation.

RESULTS OF THE EXPERIMENTS

We summarize the results of our experiments in the following tables. Table I contains results on δ, R, $\bar{R}$, ρ, whether the reconstruction resolves the original, and time, t. Table II contains $\epsilon(0)$, $\epsilon(1)$, $\epsilon(2)$, $\epsilon(3)$, $\epsilon(4)$ and $\epsilon(5)$. Table III contains $l(0.01)$, $l(0.02)$, $l(0.05)$, $l(0.10)$, $l(0.25)$ and $l(0.50)$. These are marked as 1 percent, 2 percent, 5 percent, 10 percent, 25 percent, and 50 percent in the table.

In all the tables, the numbers 4, 8, 30 and 6 on the top refer to the number of projections. The first three sets are equally spaced, while the last set consists of the angles 35°, 44°, 62°, 80°, 98° and 116°.

The vertical arrangement of the four numbers in each of the entries of the table refer to the result by the summation method, followed by the results by ART after 1, 5 and 25 iterations.

As can be seen from these tables, if the number of projections is small, one iteration of ART seems to perform somewhat worse than the summation method. As the number of projections increases, the relative performance of one iteration of ART as opposed to the summation method improves, and eventually it produces superior results. The break-even point seems to be at eight projections. However, by repeated iterations, the performance of ART can be improved and it eventually surpasses the summation method in all our experiments. The improvement in ART is especially impressive with eight or more projections. In such cases, we obtain, with five iterations of ART, results which are far superior to the results of the summation method. These conclusions are clearly demonstrated on the plates showing the reconstructed pictures.

Plate B contains reconstructions of the wrench (A1), Plate C contains reconstructions of $\rho(R)$ (A2), Plate D contains reconstructions of Judy (A3), and Plate E contains reconstructions of the stomata (A4). Plates F and G contain the contoured versions of the reconstructions of the wrench and $\rho(R)$.

TABLE I

	WRENCH (A1)				ρ (R) (A2)				JUDY (A3)				STOMATA (A4)			
	4	8	30	6	4	8	30	6	4	8	30	6	4	8	30	6
δ	.21	.19	.18	.26	.24	.23	.21	.24	.16	.15	.14	.23	.19	.18	.17	.26
	.22	.15	.13	.27	.24	.21	.13	.24	.17	.13	.10	.22	.20	.17	.11	.26
	.17	.07	.02	.21	.21	.15	.03	.19	.14	.10	.03	.20	.18	.13	.03	.23
	.11	.03	.00	.15	.19	.10	.00	.15	.13	.10	.03	.19	.18	.12	.03	.22
R	.60	.50	.48	.91	1.1	.99	.87	1.1	.32	.30	.28	.54	.42	.40	.37	.62
	.82	.52	.35	1.1	1.2	1.0	.54	1.2	.41	.32	.23	.54	.48	.40	.25	.63
	.49	.16	.03	.69	.91	.56	.08	.79	.29	.20	.07	.45	.40	.28	.06	.55
	.26	.05	.00	.41	.67	.29	.00	.49	.27	.19	.06	.42	.38	.25	.05	.51
$\bar{R}$	.03	.05	.05	.08	.08	.07	.04	.09	.11	.10	.10	.20	.17	.15	.15	.22
	.06	.01	.01	.09	.08	.05	.01	.09	.16	.10	.06	.19	.17	.13	.05	.21
	.02	.00	.00	.05	.05	.02	.00	.04	.12	.08	.03	.19	.16	.10	.01	.18
	.01	.00	.00	.02	.04	.01	.00	.02	.11	.07	.02	.19	.16	.10	.01	.19
ρ	24	24	23	25	22	22	21	23	15	17	17	18	14	17	20	19
	19	17	15	20	15	15	12	17	15	12	13	14	13	13	14	18
	7.3	3.6	1.2	8.3	5.9	5.0	1.5	6.3	2.9	1.6	.87	3.1	2.5	2.2	.88	4.0
	2.8	.92	.01	2.8	1.8	1.8	.03	2.0	.54	.35	.16	.94	.42	.52	.12	1.0
resolve?	Yes	Yes	Yes	No	No	No	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes	No
	Yes	Yes	Yes	No	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
t	1.8	3.4	13	3.4	1.7	3.4	13	3.4	1.7	3.6	14	3.5	1.8	3.6	14	3.5
	1.6	3.4	14	3.3	1.6	3.4	14	3.3	1.6	3.4	14	3.3	1.6	3.4	14	3.3
	8.0	17	68	17	8.0	17	68	17	8.0	17	68	17	8.0	17	68	17
	40	86	340	83	40	86	340	83	40	86	340	83	40	86	340	83

In all plates, the first row gives the reconstructions based on 4 equally spaced projections, the second row gives the reconstructions based on 8 equally spaced projections, the third row gives the reconstructions based on 30 equally spaced projections and the fourth row gives the reconstructions based on the 6 projections from the 81° range. Within each row, the first picture is the result obtained by the summation method, which is followed by the results obtained by ART after 1, 5 and 25 iterations, respectively.

There are a number of observations worth making regarding the results of our experiments.

First of all, the basic measures δ and R give, with hardly any exceptions, the same ordering between different experiments for the same picture. There seems to be little reason to use the one rather than the other. The orderings based on these measures are also in very strong correspondence with the orderings based on any of the resolution measures.

Another interesting point is that there is no instance when a picture reconstructed by ART does not resolve the original after five iterations.

The third observation is in reference to Table III. Very little information can be gained from a digitization with fewer than 8×8 points. In all our experiments the summation method produced a picture whose 8×8 version differs from the 8×8 original at some point by at least 0.1. In fact, for the wrench (A1), even 30 projections can only produce a picture, where the difference between the 8×8 pictures is more than .25, at least at one point.

It has been claimed for ART (Frieder & Herman [1971]) that for an $n \times n$ resolution it requires approximately n equally spaced projections. Table III bears

TABLE II

	WRENCH (A1)				ρ (R) (A2)				JUDY (A3)				STOMATA (A4)			
	4	8	30	6	4	8	30	6	4	8	30	6	4	8	30	6
ϵ (0)	.84	.79	.75	.81	.82	.68	.62	.83	.55	.48	.48	.75	.64	.57	.55	.93
	.75	.67	.90	.98	.99	.99	.77	.94	.58	.61	.65	.78	.75	.65	.65	.91
	.81	.54	.21	.95	1.0	.97	.25	.97	.56	.46	.20	.69	.74	.55	.15	.83
	.85	.35	.00	.80	1.0	.93	.01	.99	.58	.46	.17	.71	.71	.54	.13	.87
ϵ (1)	.66	.61	.64	.74	.72	.59	.55	.75	.53	.43	.42	.70	.58	.52	.48	.73
	.68	.55	.64	.87	.90	.80	.56	.79	.51	.54	.50	.58	.68	.53	.46	.69
	.57	.26	.08	.74	.87	.53	.08	.71	.51	.38	.08	.64	.63	.46	.08	.64
	.50	.12	.00	.65	.92	.39	.00	.62	.53	.37	.07	.64	.59	.47	.06	.65
ϵ (2)	.60	.55	.57	.67	.54	.53	.50	.65	.41	.36	.34	.57	.50	.44	.42	.64
	.62	.42	.35	.75	.52	.43	.24	.58	.43	.40	.31	.51	.55	.42	.30	.56
	.50	.14	.03	.64	.40	.26	.03	.49	.44	.24	.02	.50	.48	.29	.02	.48
	.36	.05	.00	.45	.42	.15	.00	.41	.47	.20	.01	.50	.45	.27	.01	.45
ϵ (3)	.33	.26	.27	.50	.23	.24	.20	.25	.25	.25	.23	.42	.19	.21	.21	.38
	.41	.24	.18	.53	.29	.21	.11	.26	.25	.19	.14	.40	.22	.20	.16	.36
	.23	.05	.01	.42	.19	.10	.01	.15	.13	.89	.01	.30	.13	.10	.01	.25
	.09	.01	.00	.23	.12	.05	.00	.09	.13	.08	.00	.29	.12	.08	.00	.17
ϵ (4)	.15	.10	.09	.31	.08	.09	.09	.16	.10	.11	.10	.11	.06	.05	.07	.15
	.12	.09	.08	.33	.06	.06	.05	.15	.07	.04	.07	.10	.06	.05	.07	.17
	.07	.01	.01	.21	.03	.02	.00	.08	.03	.01	.01	.06	.04	.02	.01	.06
	.03	.00	.00	.09	.02	.01	.00	.03	.02	.01	.00	.08	.03	.02	.00	.03
ϵ (5)	.04	.03	.02	.02	.02	.02	.02	.03	.00	.01	.02	.05	.01	.01	.02	.07
	.07	.05	.04	.05	.02	.03	.02	.03	.01	.02	.02	.02	.02	.01	.01	.06
	.03	.01	.00	.04	.01	.01	.00	.01	.00	.00	.00	.01	.01	.00	.00	.01
	.01	.00	.00	.02	.01	.00	.00	.00	.00	.00	.00	.01	.00	.00	.00	.01

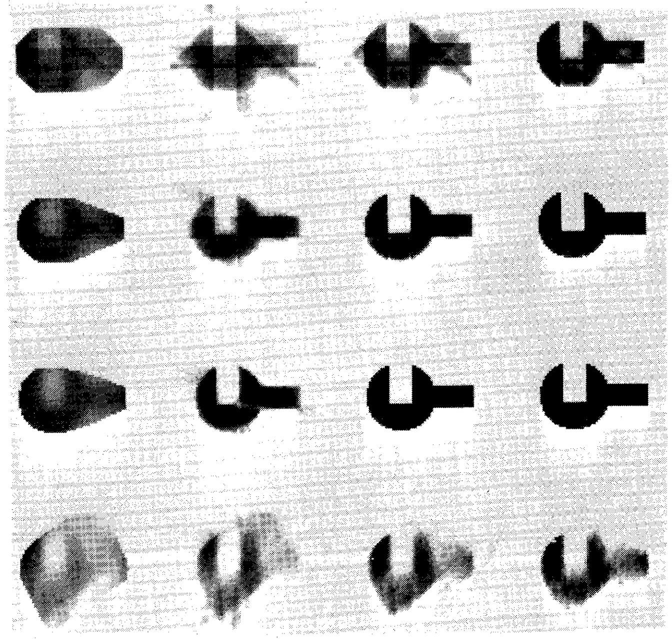


Plate B

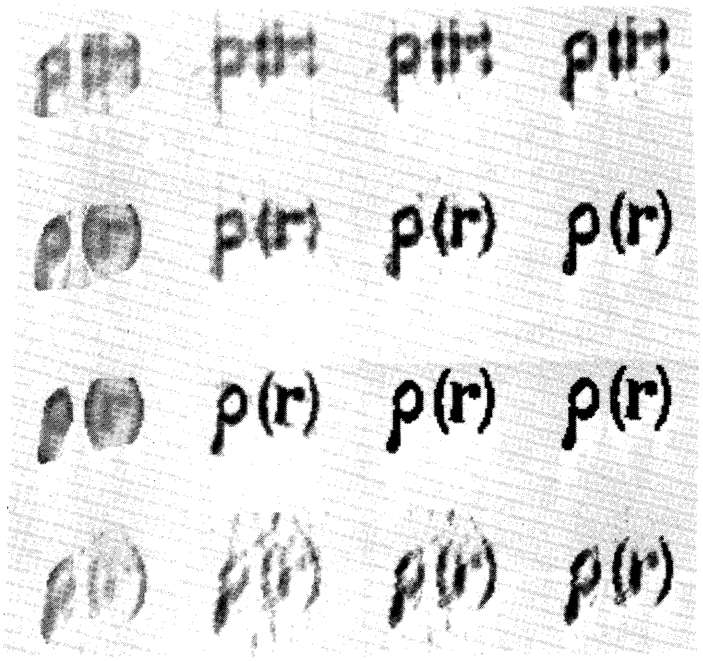


Plate C

TABLE III

	WRENCH (A1)				ρ (R)		(A2)		JUDY (A3)				STOMATA (A4)			
	4	8	30	6	4	8	30	6	4	8	30	6	4	8	30	6
1%	1	1	1	1	1	1	1	1	2	1	1	1	1	2	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	2	4	1	2	1	4	1	2	2	4	2	2	2	8	2
	1	4	64	1	2	4	32	2	2	4	8	2	2	2	8	2
2%	1	1	1	1	1	2	2	1	2	2	2	1	2	2	2	1
	1	1	1	1	1	1	1	1	2	2	1	1	1	2	2	1
	1	4	8	1	2	2	8	2	2	4	16	2	2	4	16	2
	2	8	64	2	2	4	64	2	4	4	16	2	2	4	16	2
5%	2	2	2	2	2	2	2	2	2	2	2	1	2	2	2	1
	1	2	2	1	2	2	4	2	2	4	2	2	2	2	2	1
	2	8	16	2	4	4	16	2	4	4	16	2	4	4	16	2
	4	8	64	2	4	8	64	4	4	4	16	2	4	4	16	4
10%	2	2	4	2	4	4	4	2	2	2	2	2	4	4	4	2
	2	4	4	2	4	4	4	2	4	4	4	4	4	4	4	2
	4	8	32	2	4	8	32	4	4	8	32	4	4	4	32	4
	8	16	64	4	4	8	64	8	4	8	32	4	8	8	32	4
25%	4	4	4	2	8	8	8	4	8	4	8	4	8	8	8	4
	4	8	8	2	4	8	16	4	4	8	8	4	8	8	8	4
	8	16	64	4	8	8	32	8	8	16	64	4	8	8	64	4
	8	32	64	8	8	16	64	8	8	16	64	4	8	8	64	8
50%	8	8	8	4	8	8	16	8	16	64	64	8	16	16	32	8
	8	16	16	4	8	16	16	8	16	16	16	8	8	16	32	8
	16	32	64	8	16	16	64	16	16	64	64	8	16	32	64	16
	32	64	64	16	16	32	64	16	16	64	64	16	16	32	64	16

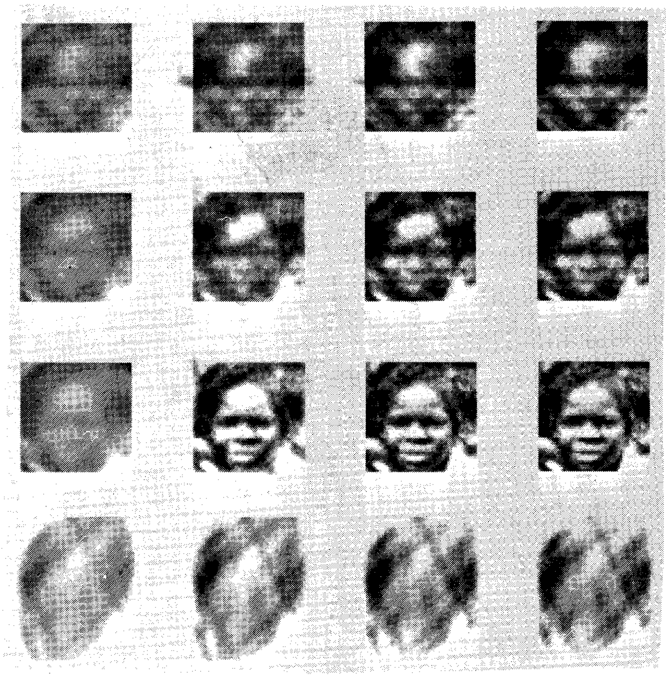


Plate D

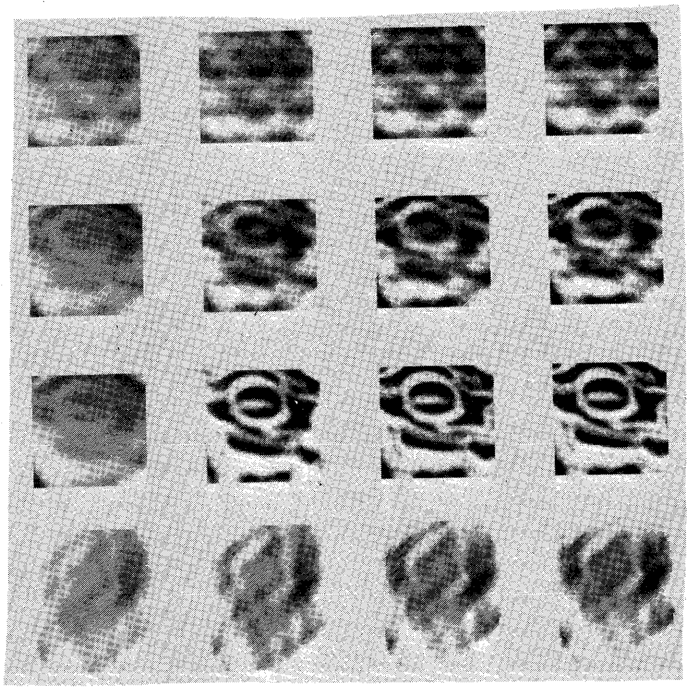


Plate E

this out. For the binary valued pictures with n equally spaced projections the error at $n \times n$ resolution is less than 0.05 at all points, while for the half-tone pictures it is less than 0.1. (This is after 25 iterations, sometimes sooner.)

For the case of 30 projections, for all four test patterns, the error at each point in the 64×64 reconstructed picture after 25 iterations of ART is less than .175, and in the case of the binary valued pictures it is actually less than .015 (see Table II).

For perfect contouring we need an error which is less than .5 at each point. Thus, we see that contoured 16×16 reconstructions of all the pictures are perfect, with all four sets of angles, when ART is used with 25 iterations. For the summation method, the same statement holds only for 4×4 reconstructions.

At two points, our results do not strictly correspond to results in earlier publications.

The first of these is the bad quality of the contoured output of the summation method (Plates F and G). The contoured output obtained by Vainshtein (1971a) in the same experiment appears to be superior. The reason for this difference is that Vainshtein contoured at a level which will make the contoured input and output to have the same density, while we made the half-tone output to have the same density as the input, and then contoured at .5. The reader can easily judge for himself, from Plates B and C, what the effect of contouring at different levels would have been.

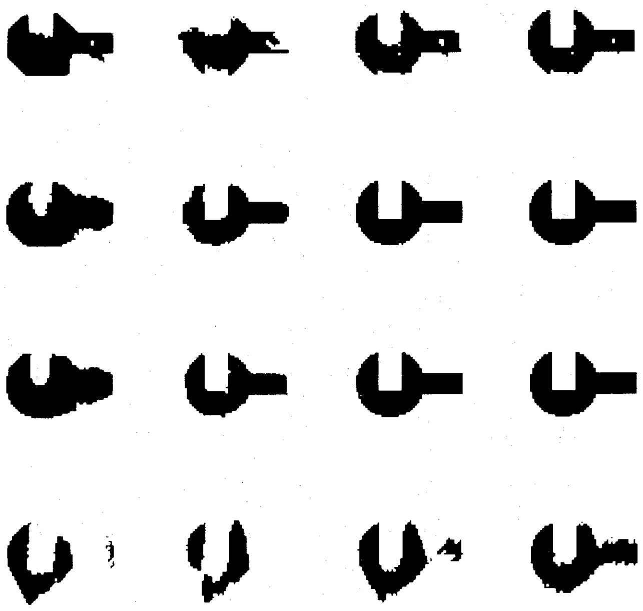


Plate F

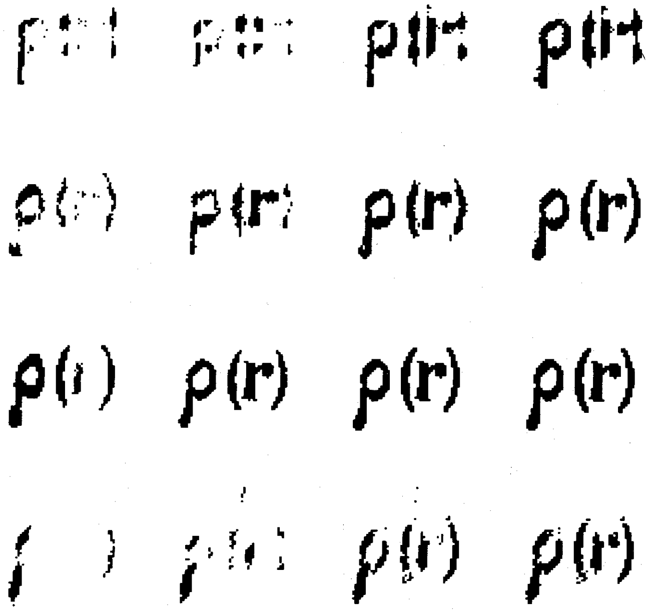


Plate G

The other point concerns the performance of ART on the half-tone pictures using 6 projections from the 81° range. The results appear to be much worse than what have previously been obtained either for Judy (Gordon, Bender & Herman [1970]) or the stomata (Herman & Rowland [1971]). There are two reasons for this. One is that we are putting a fairly wide white frame around the picture. This frame does not contribute to the total density, but the fact that it goes all the way around the picture cannot possibly be detected from projections taken in a small range. Since ART is intended to produce a picture as smooth as possible while satisfying the projections, it will smooth out the picture into that part of the original white frame which is not uniquely determined from the projections. This will also bring with itself a general lowering of density in the middle portion of the picture, causing some distortions in trying to satisfy some of the projections. The second reason for the bad quality of the reconstructions of the stomata, as opposed to earlier experiments with angles within the same range, is that previously all the projection angles clustered around the horizontal, while due to the rotation discussed in Section two, now they cluster around the vertical. The stomata seem to have many horizontal bands, thus, reconstructions from projections from a small horizontal range produce better results, than from a small vertical range. (This is not the case with Judy.) The countermeasure to being a victim of the orientation of the object relative to the small range of views is to average a number of independent reconstructions (see Herman & Rowland [1971]). The following table gives the value of δ when the Judy and the stomata test patterns are reconstructed, with and without a white band around them, with six views from two small ranges, as well as the average in each case. All results are based on ART after 25 iterations.

	Angles −45°, −36°, −18°, 0°, 18°, 36°	Angles 35°, 44°, 62°, 80°, 98°, 116°	Average
Judy 49×49 version	.14	.16	.13
Judy 64×64 version	.19	.19	.16
stomata 50×50 version	.16	.23	.17
stomata 64×64 version	.17	.22	.17

As can be seen from this table, averaging considerably improves the quality of the reconstruction, for the 64×64 pictures. Using a smaller frame can also be helpful in this respect. Thus, the bad results we obtained for half-tone pictures with six projections from a small range can be avoided by a careful choice of the square in which $f(x, y)$ is assumed to be possibly non-zero (it should be as small as possible), or by averaging. Naturally, the same comments apply to the summation method as well.

ACKNOWLEDGMENTS

This research has been supported by NSF Grant GJ998. The author is most grateful to Mr. Stuart Rowland for his constant help with programming and in other matters. The following colleagues have also been helpful both by enlightening discussions and in producing the actual pictures: R. Bender, G. Frieder, R. Gordon and J. Rowe. The pictures have been produced by the courtesy of Optronics International, Chelmsford, Massachusetts.

REFERENCES AND BIBLIOGRAPHY

S H BELLMAN R BENDER R GORDON
J E ROWE JR
ART is science, being a defense of algebraic reconstruction techniques for three-dimensional electron microscopy
J Theor Biol 32 pp 205–216 1971

R BENDER S H BELLMAN R GORDON
ART and the ribosome: a preliminary report on the three-dimensional structure of individual ribosomes determined by an algebraic reconstruction technique
J Theor Biol 29 pp 483–487 1970

S K CHANG
The reconstruction of binary patterns from their projections
Comm ACM 14 pp 21–25 1971

S K CHANG G L SHELTON
Two algorithms for multiple-view binary pattern reconstruction
IEEE Transactions on Systems Man and Cybernetics
1 pp 90–94 1971

R A CROWTHER
Procedures for three-dimensional reconstruction of spherical viruses by Fourier synthesis from electron micrographs
Phil Trans Roy Soc Lond B 261 pp 221–230 1971

R A CROWTHER L A AMOS J T FINCH
D J DEROSIER A KLUG
Three dimensional reconstructions of spherical viruses from electron micrographs
Nature 226 pp 421–425 1970

R A CROWTHER D J DEROSIER A KLUG
The reconstruction of a three-dimensional structure from projections and its application to electron microscopy
Proc Roy Soc Lond. A 317 pp 319–340 1970

R A CROWTHER A KLUG
ART and science, or, conditions for 3-D reconstruction from electron microscope images
J Theor Biol 32 199–203 1971

D J DEROSIER
Three dimensional image reconstruction of helical structures
Berichte der Bunsen-Gesellschaft 74 pp 1127–1128 1970

D J DEROSIER
Three-dimensional image reconstruction of helical structures
Phil Trans Roy Soc Lond B 261 pp 209–210 1971

D J DEROSIER
Reconstruction of three-dimensional images from electron micrographs
Biophysical Society Abstracts Fifteenth Annual Meeting February 15–18 New Orleans p 218a 1971

D J DEROSIER A KLUG
Reconstruction of three dimensional structures from electron micrographs
Nature 217 pp 130–134 1968

D J DEROSIER P B MOORE
Reconstruction of three-dimensional images from electron micrographs of structures with helical symmetry
J Mol Biol 52 pp 355–369 1970

J T FINCH A KLUG
Three-dimensional reconstruction of the stacked-disk aggregate of tobacco mosaic virus protein from electron micrographs
Phil Trans Roy Soc Lond B 261 pp 211–219 1971

G FRIEDER G T HERMAN
Resolution in reconstructing objects from electron micrographs
J Theor Biol 33 pp 189-211

N T GAARDER G T HERMAN
Algorithms for reproducing objects from their x-rays
Conferencia Internacional IEEE Mexico 1971 Oaxtepec Mexico January 19-21 1971 also to appear in Computer Graphics and Image Processing 1972

J B GARRISON D G GRANT W H GUIER
R J JOHNS
Three dimensional roentgenography
American J of Roentgenology Radium Therapy and Nuclear Medicine 105 pp 903–906 1969

R GORDON R BENDER G T HERMAN
Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and x-ray photography
J Theor Biol 29 pp 471–481 1970

R GORDON G T HERMAN
Reconstruction of pictures from their projections
Comm ACM 14 pp 759-768 1971

E G GRAY R A WILLIS
Problems of electron stereoscopy of biological tissues
J Cell Sci 3 pp 309–326 1968

J L HARRIS
Resolving power and decision theory
J of the Optical Society of America 54 pp 606–611 1964

R G HART
Electron microscopy of unstained biological material: The polytropic montage
Science 159 pp 1464–1467 1968

G T HERMAN S ROWLAND
Resolution in ART: An experimental investigation of the resolving power of an algebraic picture reconstruction technique
J Theor Biol 33 pp 213-223 1971

W HOPPE
The finity postulate and the sampling theorem of the three dimensional electron microscopical analysis of aperiodic structures
Optik 29 pp 617–621 1969

H E HUXLEY J A SPUDICH
Application of 3-D reconstruction to problems of muscle structure
Biophysical Society Abstracts Fifteenth Annual Meeting February 15–18 New Orelans p 220a 1971

A KLUG
Optical diffraction and filtering and three-dimensional reconstruction from electron micrographs
Phil Trans Roy Soc Lond B 261 pp 173–179 1971

J A LAKE H S SLAYTER
Three dimensional structure of the chromatoid body of Entamoeba invadens
Nature 227 pp 1032–1037 1970

E R MILLER E M McCURRY B HRUSKA
An infinite number of laminagrams from a finite number of radiographs
Radiology 98 pp 249–255 1971

P B MOORE H E HUXLEY D J DEROSIER
Three-dimensional reconstruction of F-actin, thin filaments and decorated thin filaments
J Mol Biol 50 pp 279–295 1970

R S MORGAN
Structure of ribosomes of chromatoid bodies: three-dimensional Fourier synthesis at low resolution
Science 162 pp 670–671 1968

G N RAMACHANDRAN
Reconstruction of substance from shadow I. mathematical theory with application to three-dimensional radiography and electron microscopy
Proc Indian Acad Sci in press 1971

G N RAMACHANDRAN
A V LAKSHMINARAYANAN
Three-dimensional reconstruction from radiographs and electron micrographs II. application of convolutions instead of Fourier transforms
Proc Natl Acad Sci US 68 pp 2236-2240 1971a

G N RAMACHANDRAN
A V LAKSHMINARAYANAN
Three-dimensional reconstruction from radiographs and electron micrographs III. description and application of the convolution method
Indian J Pure & Applied Phys Raman Memorial Issue in press 1971b

A ROSENFELD
Picture processing by computer
Academic Press Inc New York New York 1969

O J TRETIAK M EDEN W SIMON
Internal structure from x-ray images
8th International Conference on Medical and Biological Engineering Chicago 20–25 July 1969 Session 12-1 1969

B K VAINSHTEIN
Finding the structure of objects from projections
Soviet Physics—Crystallography 15 pp 781–787
Translated from Kristallografiya 15 pp 894–902 1970 1971a

B K VAINSHTEIN
The synthesis of projecting functions
Doklady Akademii Nauk SSSR 196 pp 1072–1075 1971b

M WATANABE S SASAKI N ANASAWA
Image analysis of electron micrographs by computer
JOEL News 9E1 pp 9–11 1971

APPENDIX

Justification of the use of digitized rather than continuous test patterns

A basic assumption of any picture reconstruction process is that the picture is reasonably smooth relative to the size of the ray widths. It is clearly unreasonable to expect an algorithm to reconstruct detail which is much finer than the width of the rays.

Since our digitization was done approximately on the same scale as the ray width (i.e., the distance between two adjacent points in the matrix on which the digitized picture is defined is of the same order of magnitude as the ray-widths), we may conclude that values of the digitized picture at neighboring points are highly correlated. Furthermore, by our smoothness assumption, if we divide the continuous picture into squares with centers at the points of digitization, at all points in the squares the gray value of the picture will be near to the digitized value at the center.

The way we have chosen the position and width of the rays is such that relative positioning of the squares and the rays will be as shown in Figure 2.

In the digitized version, the contribution to the indicated ray in Figure 2 will come from the value at

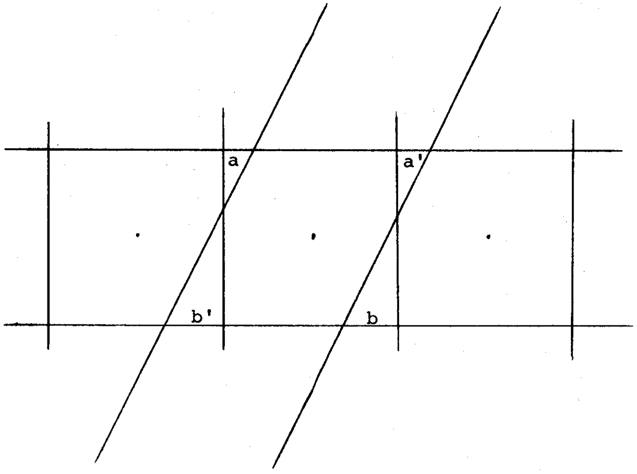


Figure 2

the center of the square, which is supposed to equal the total grayness in the square. In the continuous version, however, regions a and b in the square do not contribute to this ray, while regions a' and b', which are outside the square, do contribute. However, the area of a is the same as that of a' and area of b is the same as that of b'. Hence, if the picture function is smooth relative to the ray widths, the ray-sums of the digitized and continuous versions of the picture are approximately the same.

To demonstrate that this argument works in practice, we took ray-sums of a continuous version of the test pattern in A1, with a genuinely circular boundary. The following shows the order of differences in our measurements when we used the continuous and discrete test patterns. The discrete pattern is not binary valued, rather it is the 64×64 digitization of the continuous pattern with 16 gray levels. All results refer to the experiments with four projections, ART and five iterations.

	δ	R	ρ	$\epsilon(1)$	$\epsilon(3)$	$\epsilon(5)$
continuous test pattern	.1642	.4461	7.515	.5755	.2315	.0168
digitized test pattern	.1633	.4429	7.230	.5658	.2303	.0161

It appears that there is no essential difference between the results of the two experiments. Other tests gave similar results (see, e.g., Herman & Rowland [1971]).

Since the use of digitized test patterns makes experimentation considerably simpler, it seems advisable to use them, rather than continuous test patterns.

PRADIS—An advanced programming system for 3-D-display

by J. ENCARNACAO

Heinrich-Hertz-Institute
Berlin/Germany

and

W. GILOI

University of Minnesota
Minneapolis, Minnesota

SUMMARY

The development of PRADIS was started in early 1968 as a vehicle to implement and evaluate new schemes of man-machine-interaction, data base organization, and hidden-line detection which were being developed in our laboratory. With time it evolved into a rather elaborate programming system for generation, construction, manipulation, and perspective display of arbitrary three-dimensional objects. All relevant object types such as polyhedrons or groups of polyhedrons with convex or concave structure, objects that are described by analytical expression, as well as the most general case of objects that are bounded by arbitrarily curved surfaces can be handled. Different modes of defining and inputting objects and different-hidden-line suppression algorithms have been individually developed and implemented for these various types. Man-machine-communication has been facilitated by defining a simple command language for the user. The execution of any command can be initiated by pointing to the respective instruction in a command menu. Additionally, a keyboard (or the console typewriter) can be used for specification of parameters or the description of objects by analytical equations. Subroutines for scaling, translation, and rotation are part of the system as well as program modules which enable the user to construct three-dimensional objects by drawing two-dimensional views of them.

PRADIS is a modular system designed for implementation on a small to medium scale computer (our system has only 16 K of 24-bit words). Each module of the programming package is a functional entity, linked to other modules by an overlay technique. In the paper, the construction of the programming system and its modules, the different input modes, and the various hidden-line detection procedures are described. Representative pictures illustrate the performance of the system and its intrinsic procedures.

INTRODUCTION

PRADIS is a programming package written in FORTRAN II for the SDS 930 computer. As our configuration has only 16 K of core memory, the whole system has been divided into 16 segments (links). Each segment is a self-sustained entity that takes care of a certain complex of the total set of functions. The various subsystems exchange data via a shared common area.[1]

The first module is a control module that interprets the command language by which a user defines what class of objects he is going to deal with and what input mode he will use. The control module then calls the respective link which provides a data input in the required mode, generates objects, and executes instructions in an interactive mode. By setting certain sense switches, the user finally terminates the job or calls the control module again for further execution of other jobs. Figure 1 gives a simplified block diagram of the entire system which we will discuss more in detail in the following sections.

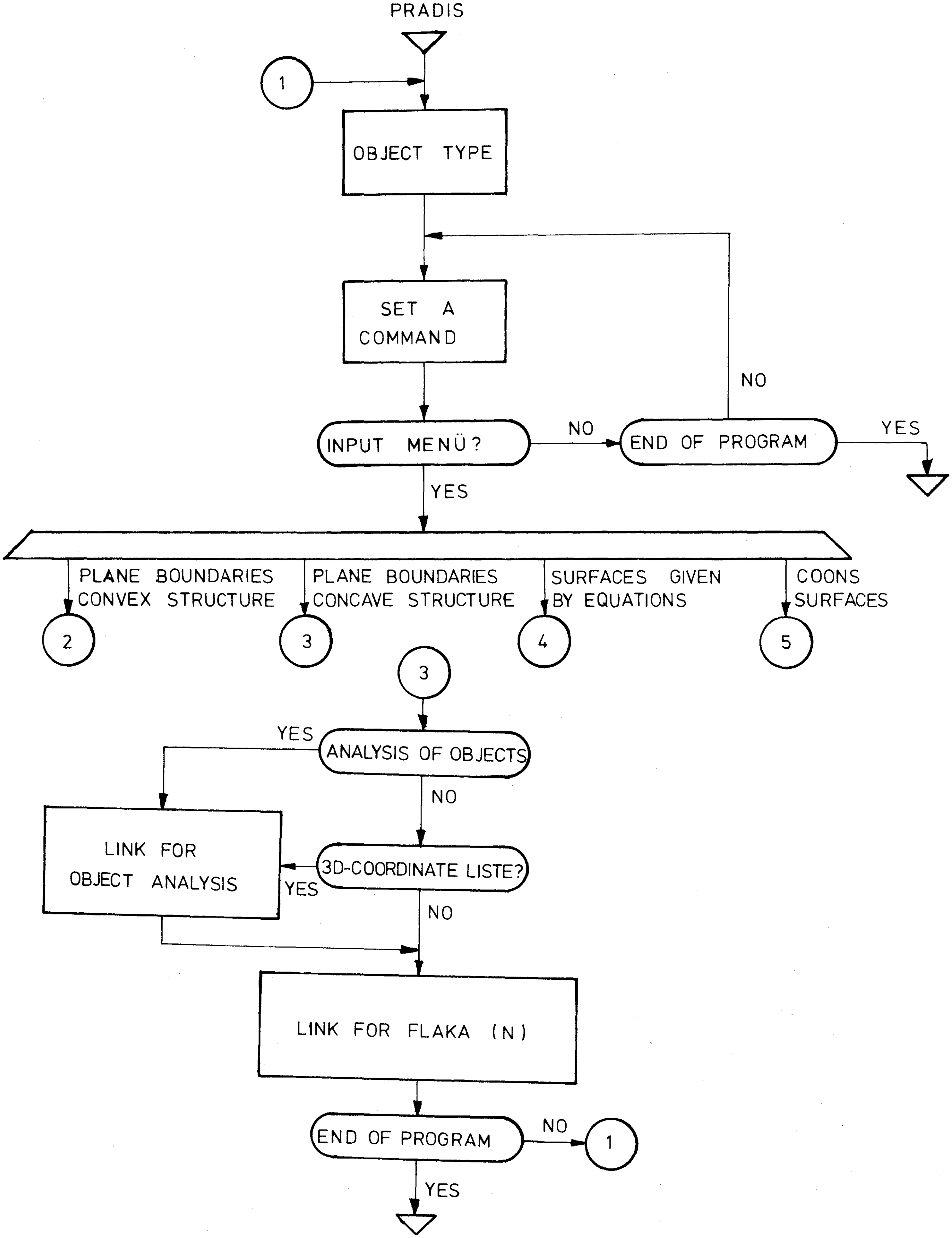


Figure 1a—Block diagram of the system PRADIS

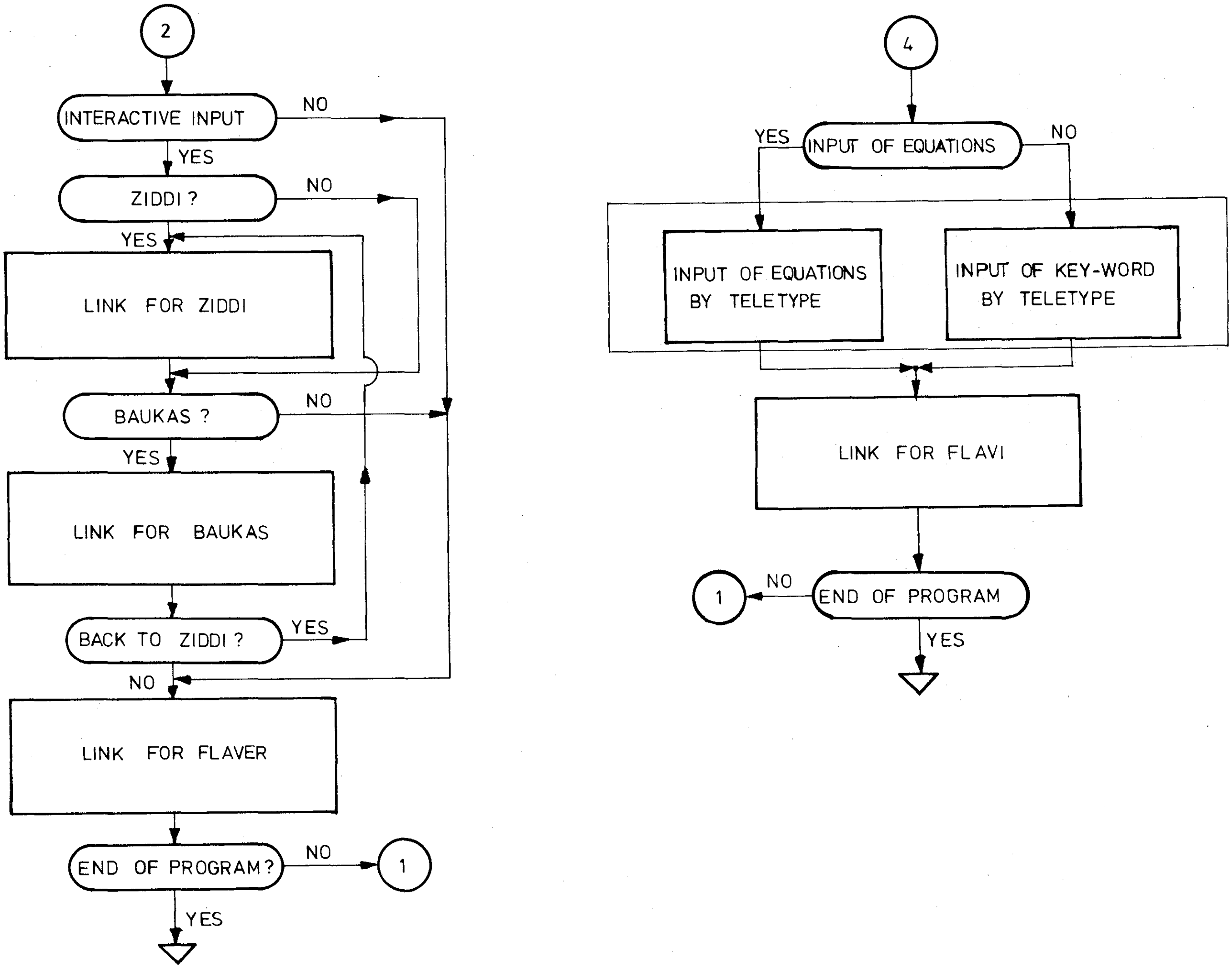


Figure 1b—Block diagram of the system PRADIS

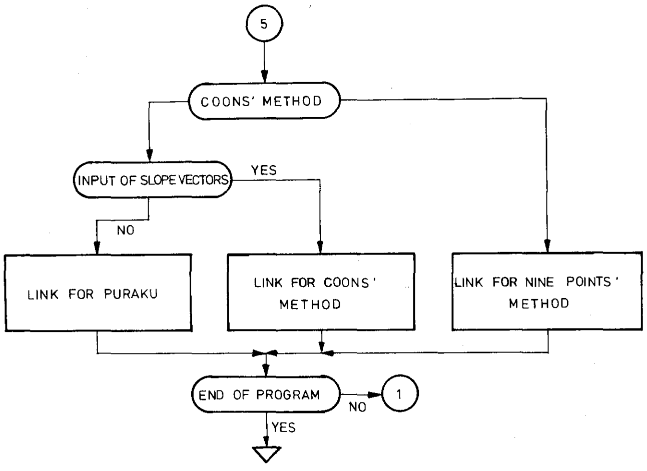


Figure 1c—Block diagram of the system PRADIS

CLASSES OF OBJECTS

The classes of objects that can be selected are:

- Groups of polyhedrons of convex structure (KOVSTR)
- Groups of polyhedrons of concave structure (KOKSTR)
- Groups of objects that are defined by functions of two variables which have to be given in analytical form (FUNGLE)
- Groups of objects described by functions of two variables which are "primitive functions" (i.e., they are stored under a function name in the library) (FUNAME)
- Groups of objects that are neither polyhedrons nor defined by analytical functions but that are composed by "surface patches" as defined by COONS' theory (FLADAR)

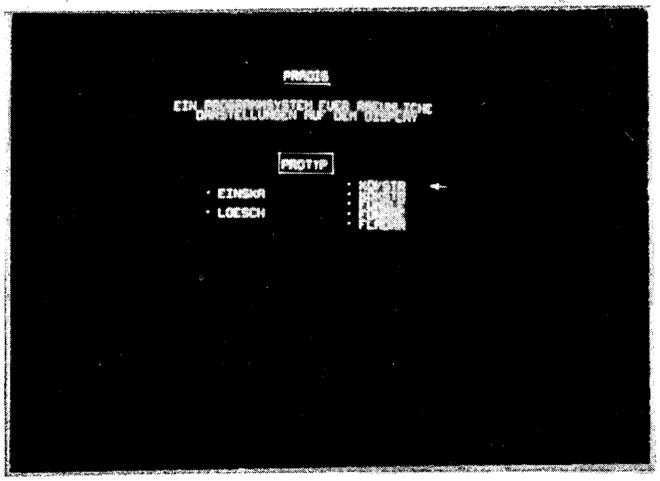


Figure 2—PROTYP—Menu

(In the present version of PRADIS 'FUNGLE' and 'FUNAME' are intrinsically the same module. However, in an extended version they have to be distinguished.)

Figure 2 shows the "menu" or "light-button field" which enables the user to select the required module by pointing first to the PROTYP command and then to a selection instruction. The selected problem type (and the respective module) is indicated by an arrow. An erroneous selection can be corrected by a delete instruction (LOESCH). Otherwise, the next instruction menu is called, depending on the object class the user wants to deal with.

MODES OF INPUT

Each module provides several input modes for the generation and manipulation of objects as shown in Table 1.

Some comments shall be given. ZIDDI and BAUKAS are able to call each other to facilitate the interactive procedure of constructing objects of convex structure. ANKOR, so to speak, is a sort of preprocessor which calls automatically LIDREI after the analysis of the inputted objects has been performed (of course, this step can be skipped by entering LIDREI directly). LIDREI includes as well as BASTRU the 4×4-matrix operation that takes care of scaling, perspective display, and rotation of objects or groups of objects. In either case the hidden lines are evaluated and suppressed. Naturally, the employed hidden-line detection strategies have to be different.

In the case of objects defined by equations the input procedure consists in typing in either the defining equation (FUNGLE) or the functions name (FUNAME).

For the fourth class of objects (i.e., general three-dimensional objects composed by surface patches) we have two optional modes of object definition and input: FLEIN 1 and FLEIN 2. FLEIN 1 provides the definition of general three-dimensional objects following COONS' method; i.e., by putting in the characteristic nodes and slope vectors.

FLEIN 2 is based on the input of nine points for a surface patch definition, providing a much easier way of defining objects. This technique will be described in a later section.

CONSTRUCTION AND DRAWING OF THREE-DIMENSIONAL OBJECTS

To our knowledge, the first program which served that purpose was developed by T. JOHNSON[2] of M.I.T. in connection with the SKETCHPAD project. In this program, an object is represented by three views

TABLE I—Input Modes for the Generation and Manipulation of Objects

object type	input commands	visibility procedure	comment
KOVSTR	ZIDDI BAUKAS BASTRU	FLAVER	Display of objects bounded by plane surfaces with convex structure
KOKSTR	ANKOR DRELIS LIDREI	FLAKA	Display of objects bounded by plane surfaces with concave structure
FUNGLE FUNAME	—	FLAVI	Display of objects defined by functions of z variables or by library-functions
FLADAR	FLEIN 1 —— PURAKU TANECK FLEIN 2	FLAVI	Display of objects that are composed by 'surface patches' (Coons)

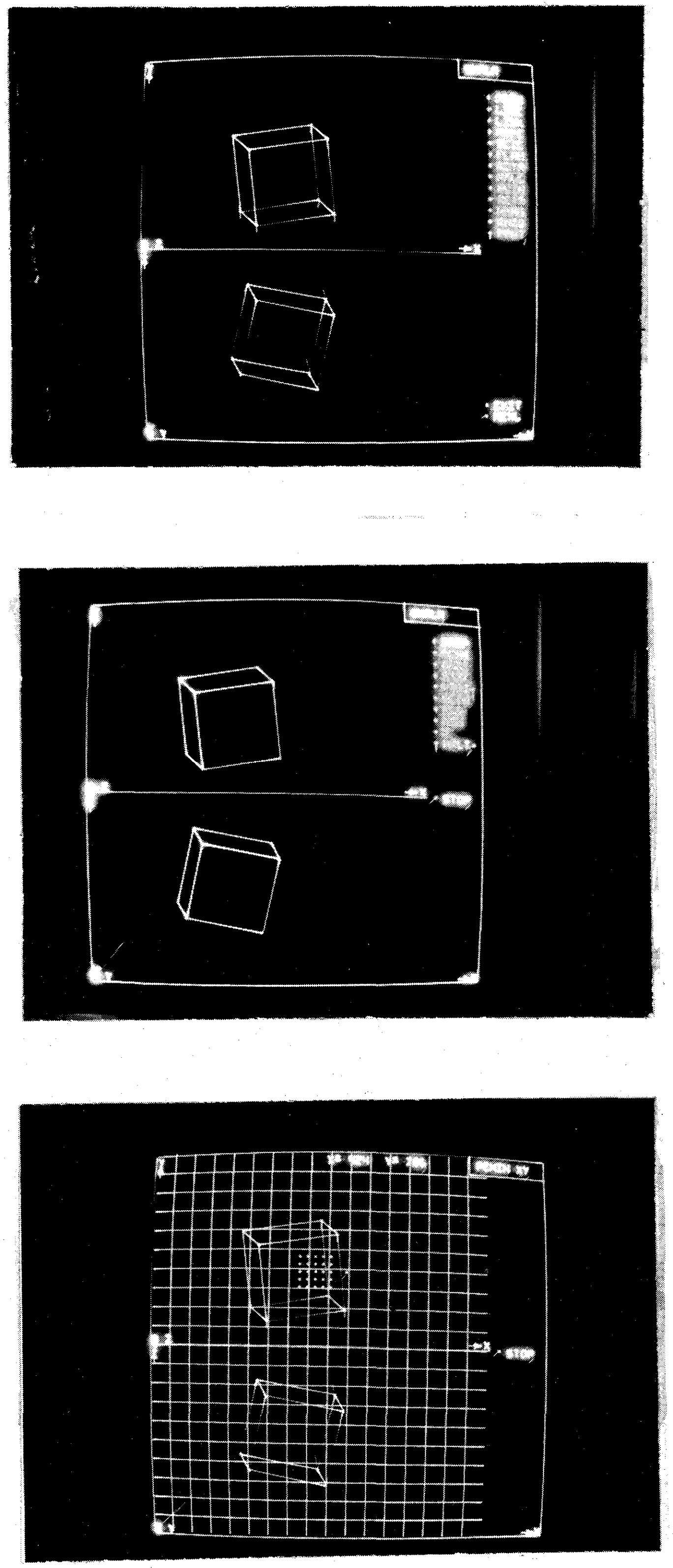


Figure 3—Examples of objects constructed with the aid of ZIDDI

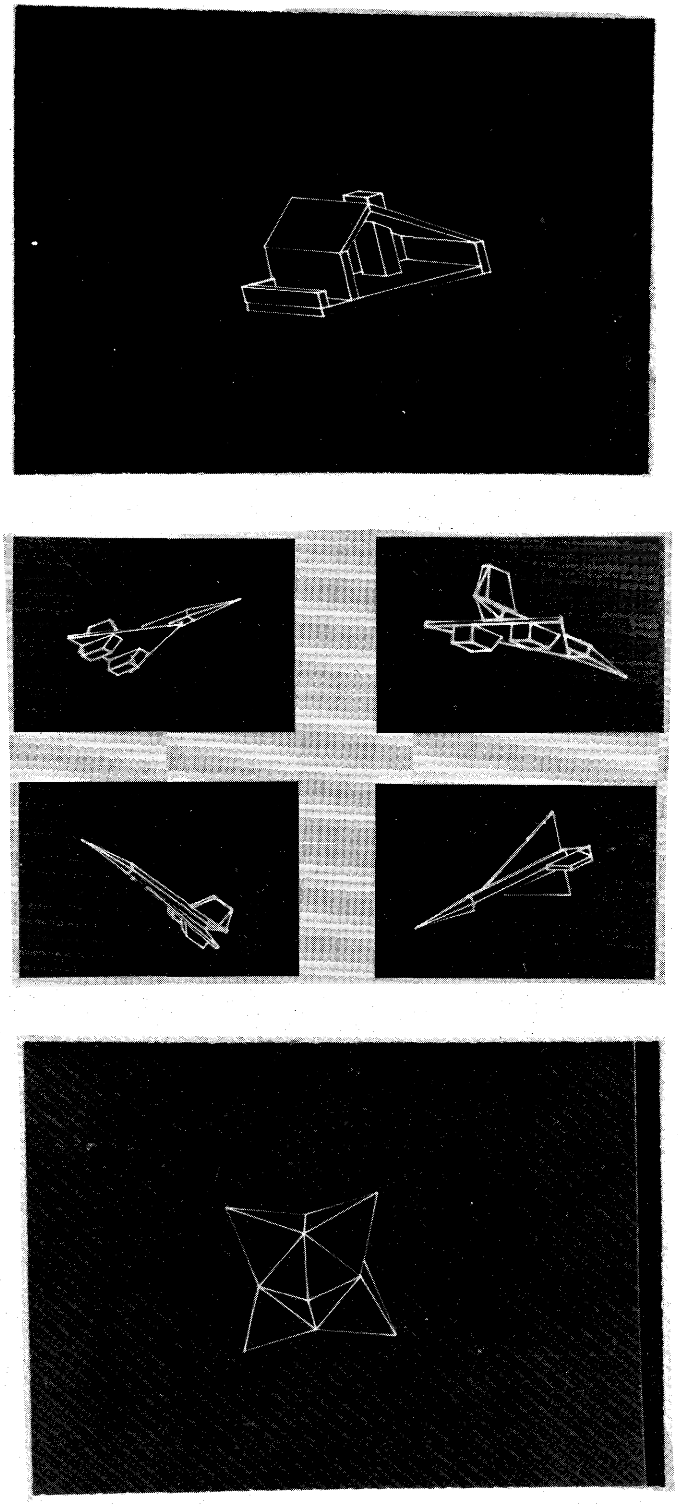
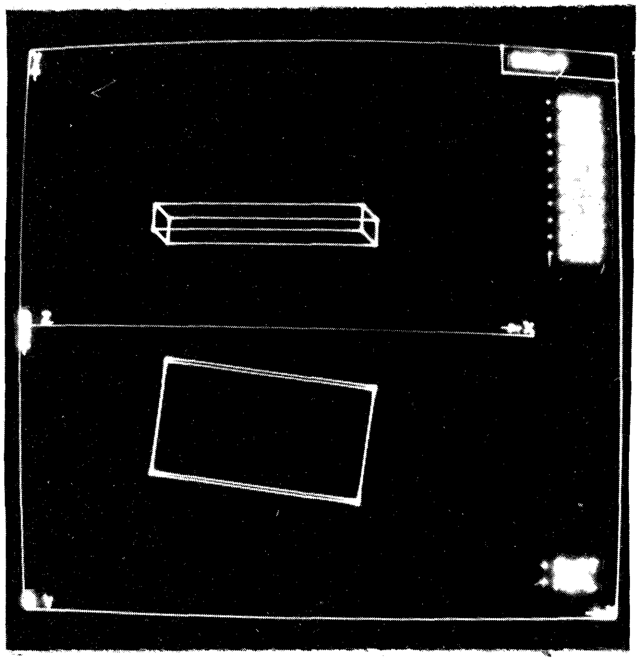


Figure 4—Examples of objects constructed with the aid of ZIDDI

and one section which have to be drawn by light-pen. Recently, two more programs have been reported,[3,4] but they do not offer the generality and flexibility of ZIDDI and BAUKAS.[5]

ZIDDI takes advantage of the man-machine-



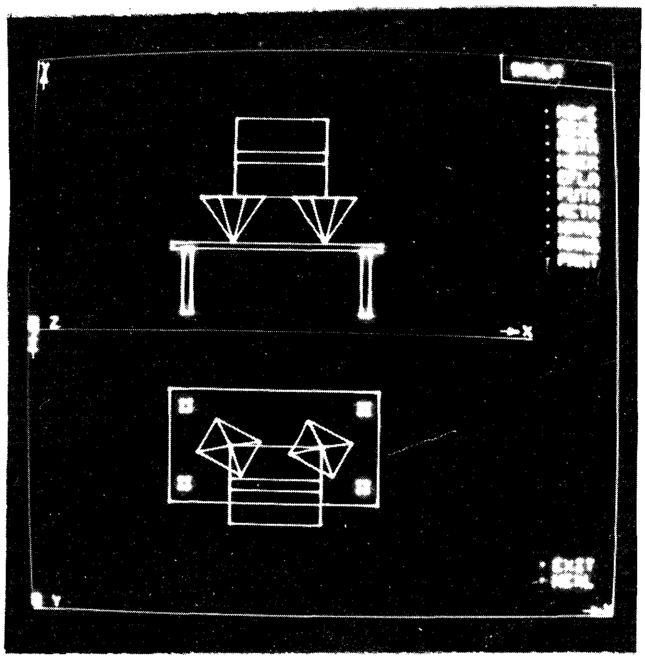


Figure 5—Illustration of BAUKAS

communication by which a display system with light-pen facilitates the task of constructing three-dimensional objects by drawing two-dimensional views. Additionally, the user has a set of powerful commands at hand for executing coordinate transforms and other operations. Using the light-pen, the user marks dots which are subsequently connected by lines. These lines may constitute surfaces which, again, bound bodies. Dots and lines can be arbitrarily added or erased. All coordinate transform procedures (scaling, translation, rotation) can be called by pointing to the respective command in the command menu, and they may be arbitrarily used to generate and modify figures and objects. In space, a point is defined by three Cartesian coordinates x, y, z. However, on the two-dimensional screen of the scope we have only two coordinates. This discrepancy is overcome by defining two areas on the screen: the xy-plane and the xz-plane. Both planes are automatically connected by linking related coordinates (for better accuracy the user can ask the system to display only one of the two planes in double size and to draw an auxiliary grid).

Concave structures are not always uniquely defined by two views, but by rotating them into an appropriate position a unique object definition can be obtained. Some simpler examples of objects that have been constructed with the aid of ZIDDI are shown in Figures 3 and 4.

BAUKAS (from German "Baukasten" = building block) provides the possibility to call primitives which are stored in a font and use them as components of a more complex object. The primitives have been constructed at an earlier time by the aid of ZIDDI. Figure 5 illustrates the way BAUKAS works. A table is going

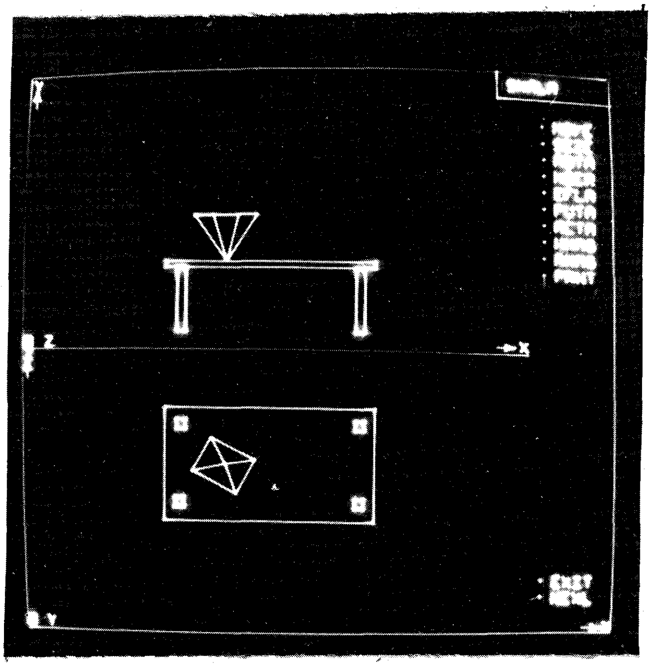


Figure 5 (cont'd)

to be constructed by using cubes as primitives. By deforming the cubes in one way, we get the table-top; by deforming it in a different way, we get the legs. On top of the table we put two other primitives: the pyramids and the cube on top of them. The flexibility of BAUKAS is illustrated by some examples given in Figure 6.

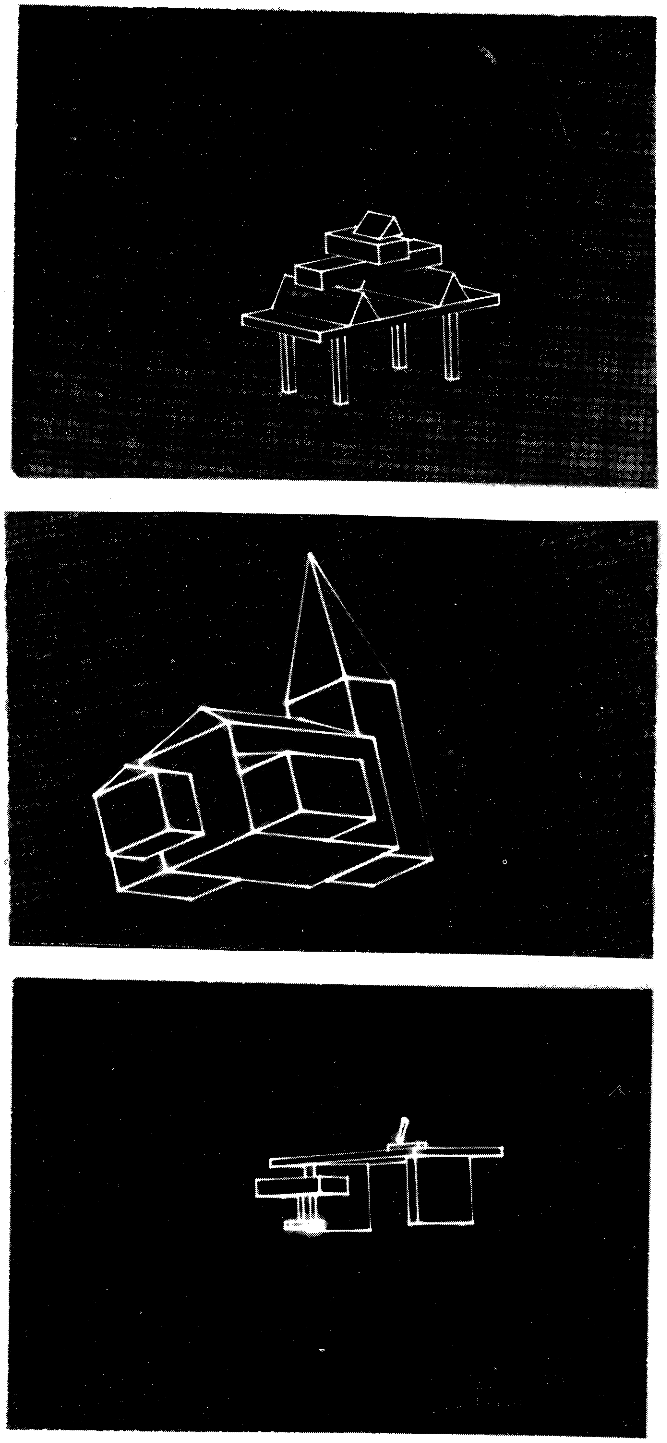


Figure 6—Examples of objects constructed with the aid of BAUKAS

ANALYSIS OF POLYHEDRONS WITH CONCAVE STRUCTURE

Concave objects are those which are bounded by one or more concave surfaces or which have cavities. A group of disconnected objects (with empty space in between) is—as a group—concave, too. It is evident, that in contrast to convex structures some difficulties may arise with respect to establishing unique relations between all points, lines, surfaces, and objects of a structure. However, these difficulties are overcome by the following procedure.[6] Prerequisite for it is, that a list is available which gives the three spatial coordinates of all points, and which marks all connections between them. Such a list may have been put in by the user or it will be generated by the program.

The most general case is that the user has three drawings which show three different (two-dimensional) views of the object. In that case the user has only to provide for each drawing a list of the two coordinate values of all relevant points, together with a second list of all visible connections. The order in which the points are numbered is arbitrary. From these lists the program generates a list of all points in space in the following way: the y-coordinate of a point in the xy-list, for example, is taken, and the yz-list is searched for the same y-coordinate. If the search is successful, the xz-list is searched for a point, whose x- and z-coordinates correspond respectively with the x-coordinate of the point taken from the xy-list and the z-coordinate of the point taken from the yz-list. Triples of x-, y-, and z-values obtained in such a way constitute the coordinates of a (spatial) point of the three-dimensional object. Eventually, the list of all spatial points is sorted with respect to increasing values of x and a running index is assigned to each triple.

The second task is to generate a list of all lines of the three-dimensional objects from the input of their two-dimensional views. A similar search procedure as in the case of the generation of the lists of spatial points is used, i.e., the three lists which belong to the two-dimensional views are searched for certain equivalences. The difference is that now an equivalence is not given by the coordinate values but by their running index. The point is that first of all the arbitrary numbering in the list of all points of the two-dimensional views has to be replaced by the corresponding running index taken from the ordered list of spatial points. In order to make that feasible, the lists of the points of the two-dimen-

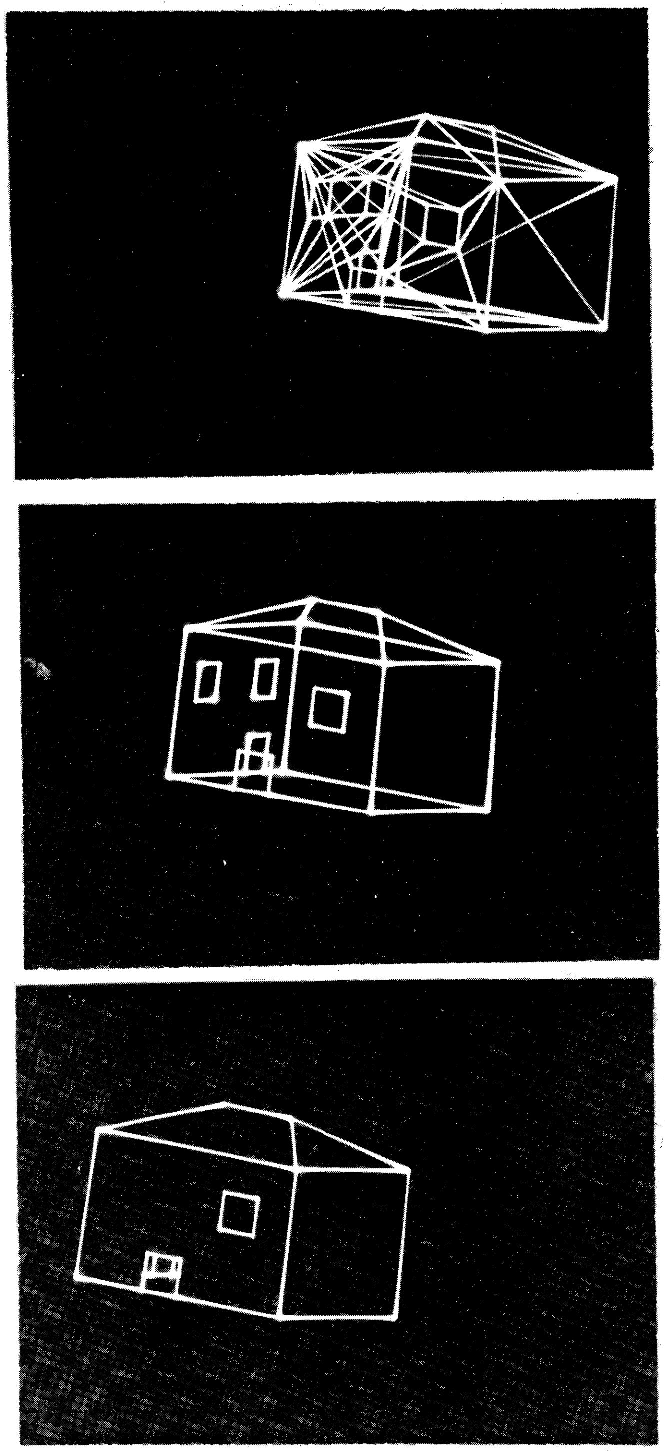


Figure 7—Examples for the results of the object analysis

sional views must at first be augmented by all connections which are existing in the three-dimensional case but not visible in all the two-dimensional views. This procedure is fairly complicated so that it would lead too far afield to explain it in detail. The next figures show some of the results we obtained with this procedure (Figures 7 to 9).

DEFINITION OF OBJECTS BY ANALYTICAL FUNCTIONS

There are three different ways of defining three-dimensional objects by analytical expressions:

(1) the implicit form $F(x, y, z) = 0$
(2) the explicit form $z = f(x, y)$
(3) the parametric form $x = f_1(u, v)$, $y = f_2(u, v)$, $z = f_3(u, v)$.

Though the implicit form is the most simple one, it does not lead to an efficient computer algorithm. The explicit form, on the other hand, does not provide a unique description (e.g., for $z = 0$ we have $f(x, y) = 0$ which does not only define all the points of a plane but all the points on a cylinder). So we have to rely on the parametric form.

In order to facilitate the input procedure, the system enables the user to key-in his equations on the display keyboard (or console typewriter, respectively). After having the equations analyzed and transformed, the system calls an appropriate algorithm for function evaluation. Furthermore, the user will be asked to specify all necessary reference coordinates, scaling factors, and other parameters. Syntactical errors will be detected, and error messages will be output. The respective language which we use is simple; its syntax has been described elsewhere.[7,8]

The typed-in equations are processed in the following steps:

(1) loading into a buffer storage area
(2) equation analysis and coding
(3) assignment of values to the parameteric variables (u, v)
(4) assignment of values to the coordinate variables (x, y, z)
(5) calculation of the right-hand expressions and storage of the obtained results.

In the second step the equations are converted into polish notation, and a push-down stack is set up. An operation matrix defines whether or not an actual operation can be immediately performed on two subsequent variables. The first row denotes the actual operators and the first column their predecessors. The meaning of the numbers in the matrix is

1: Operation cannot be executed. Bring operator into the operator stack.

2: Both operators have same priority. Execute operations.

3: Actual operator has higher priority than predecessor. Execute operation.

4: Erase parentheses (or brackets). If the expression between parentheses (brackets) is a subroutine for function evaluation, it is executed.

5: Recognition of a terminal operator.

6: Invalid combination of operators. Output error message.

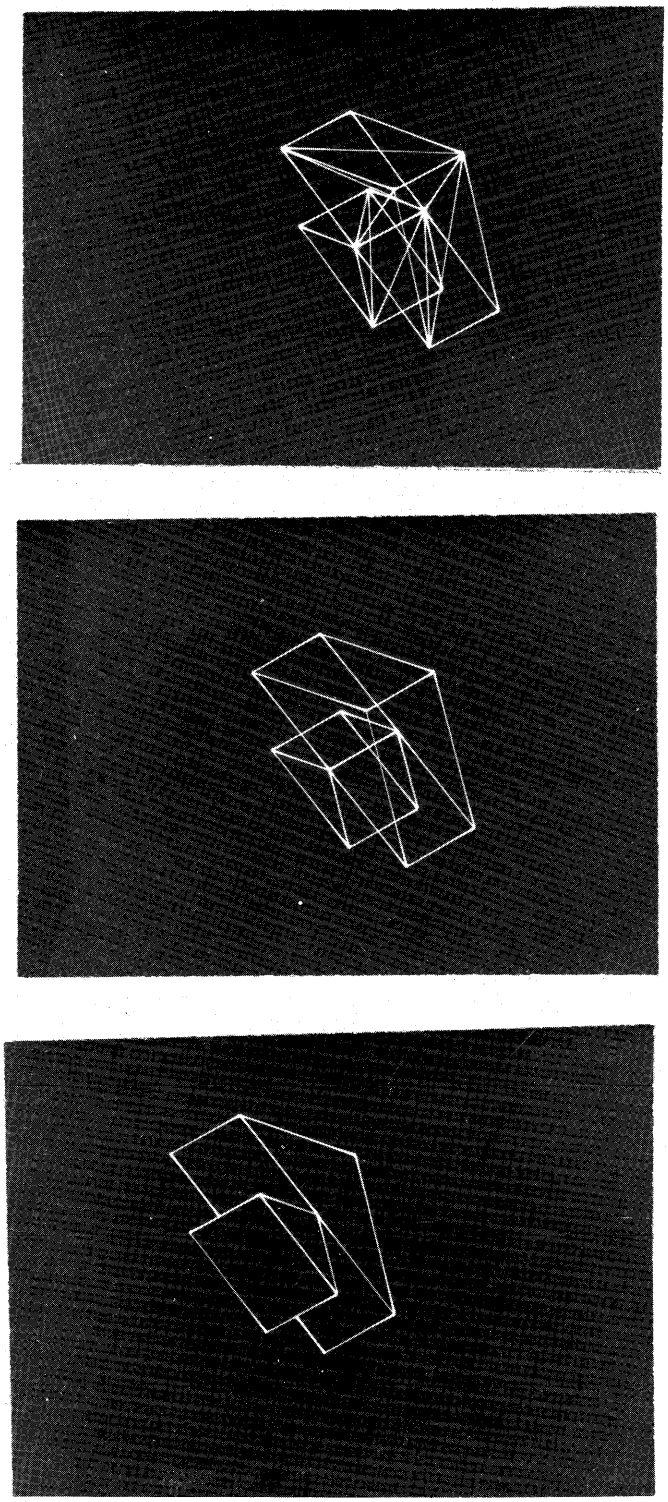


Figure 8—Examples for the results of the object analysis

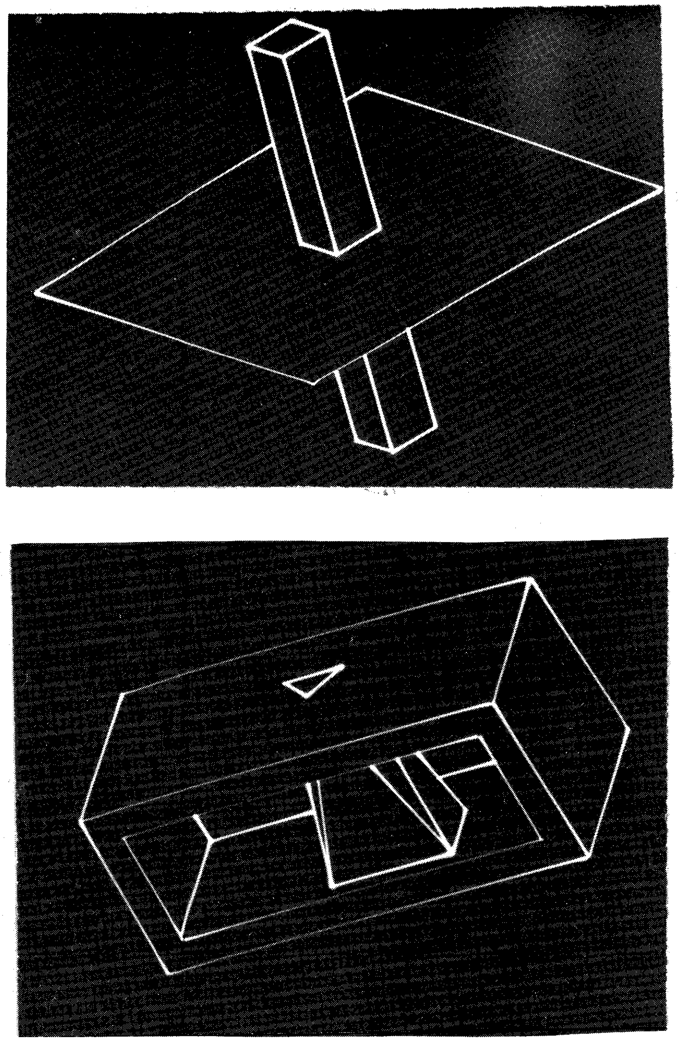


Figure 9—Examples for the results of the object analysis

OPERATION MATRIX

	+	–	*	/	(	[	)	]	A	△	‡
+	2	2	1	1	1	1	3	3	1	3	1
–	2	2	1	1	1	1	3	3	1	3	1
*	3	3	2	2	1	1	3	3	1	3	1
/	3	3	2	2	1	1	3	3	1	3	1
(	1	1	1	1	1	1	4	4	1	6	1
[	1	1	1	1	1	1	4	4	1	6	1
A	6	6	6	6	1	1	6	6	6	6	6
Blank	1	1	1	1	1	1	6	6	1	5	1
‡	3	3	3	3	1	1	3	3	1	3	2

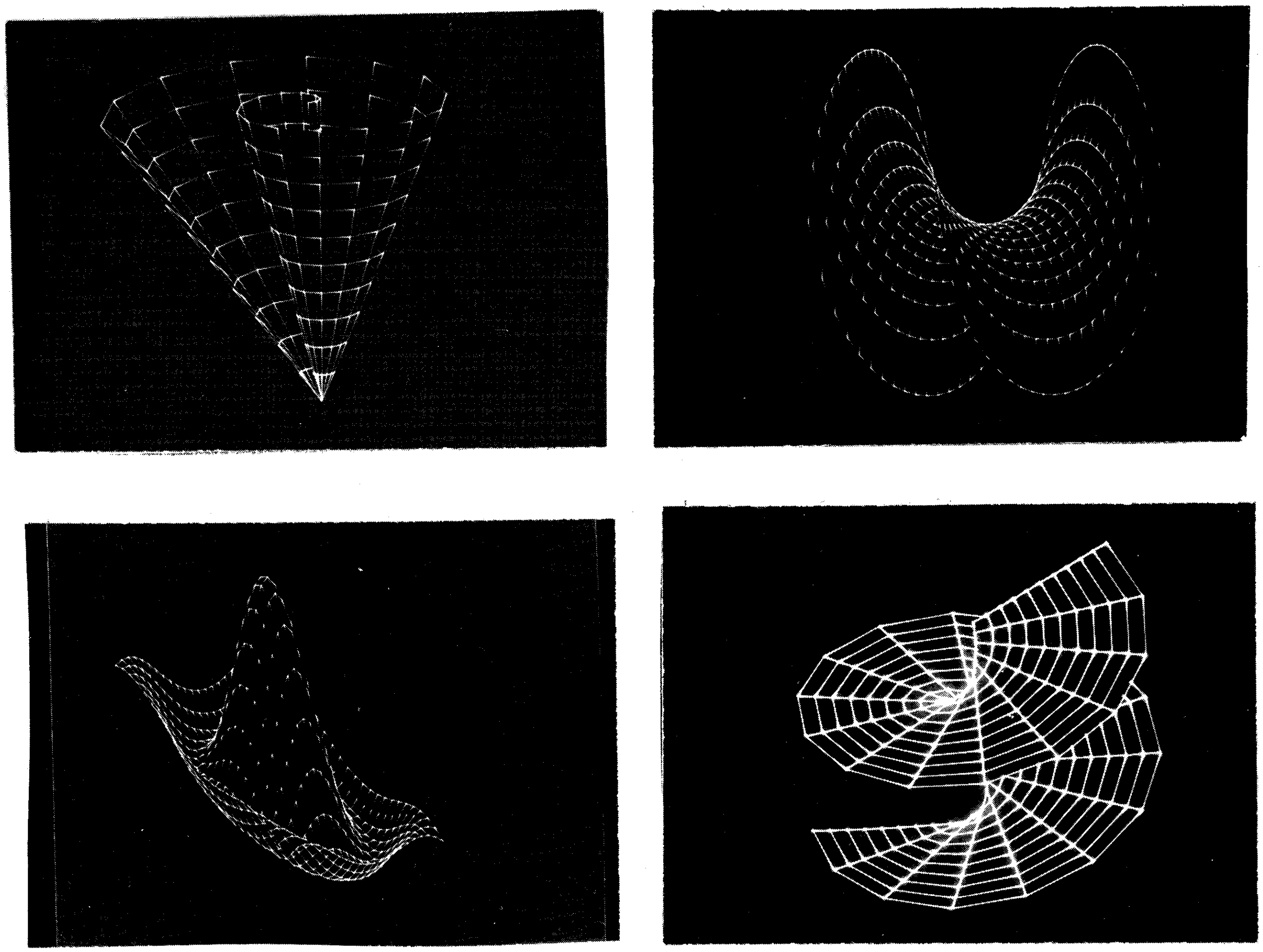


Figure 10—Surfaces generated and displayed with FLAVI

The above outlined analysis program has been written in FORTRAN II which, however, had to be supplemented by bit and character manipulation subroutines. The submodule FLAVI can be applied for hidden line detection and suppression to the thus calculated functions. Figure 10 shows some objects generated in that way.

GENERAL OBJECTS

According to COONS' theory and the nine point modification we have three main modes of defining bounding surfaces of general three-dimensional objects:

(1) The boundary contours have to be specified; blending functions (functions that define how the boundaries are connected by curved surfaces) are defined once and for all.

(2) Four corner points and eight corresponding slope vectors have to be specified for each surface patch; fixed blending functions are given, as in the first method.

(3) For each surface patch nine characteristic points have to be specified. No blending functions are required.

Generation of boundaries by polynomial approximation (FLEIN 1, PURAKU)

The first approach offers the advantage of dealing with larger entities than the "surface patches" in the second and third approach. Each boundary is approximated as an entity by a polynomial approximation. For each contour $n+1$ quintuples have to be specified; n being the degree of the approximation polynomial.

The five values of each quintuple give the three coordinates of a point through which the polynomial passes as well as two parameters for the specification of the two degrees of freedom of the surface. Thus, the degree of partitioning the objects can be lower.

However, the method shows some deficiencies. (One serious deficiency is, for example, that it is not very well suited for an interactive use of the display system but requires rather elaborate preliminary work.) It is left to the user to define the degree of each approximation polynomial; if he is not very familiar with the performance of polynomial approximations, large errors may occur. Of course, the best performance is obtained if the boundaries are conic sections.

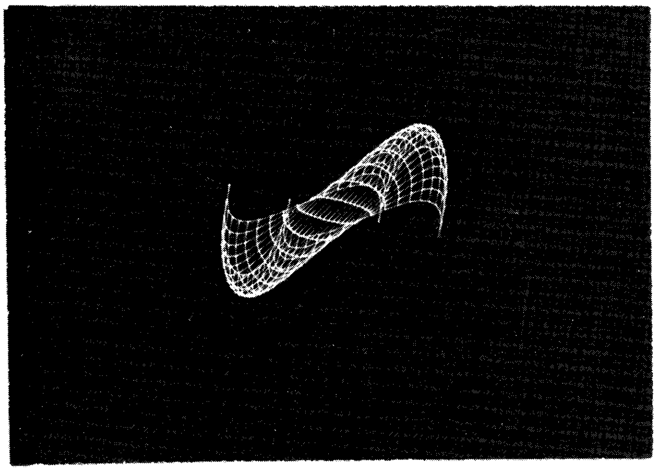


Figure 11—Object defined by COONS' method

COONS' method (FLEIN 1, TANECK)

As mentioned before, for each construction element called "surface patch" of a curved surface we have to specify four corner points and eight corresponding slope vectors, two for each boundary contour. These vectors, of course, are defined in space by specifying their components.[10] In FLEIN 1, the intrinsic blending functions are of third order. Figure 11 shows an example for an object defined in such a way. In order to obtain a smooth representation of an object, it is necessary that the user develops a good feeling of how to specify slope vectors in an optimum way. Hence, the application of this method requires a lot of experience and preliminary work with pencil and paper.

The nine point method (FLEIN 2)

Here, a surface patch is defined by specifying nine characteristic points: the four corner points, four points on the boundaries (one point on each one), and one point in the center of the surface.[11] Figure 12 gives an example: the surface element is a quadrilateral, the center point is on the same level (has the same z-coordinate) as the four corner points, and the points on the boundaries are on a higher level. In Figure 13, in contrast, the center point has been raised to the level of the boundary points.

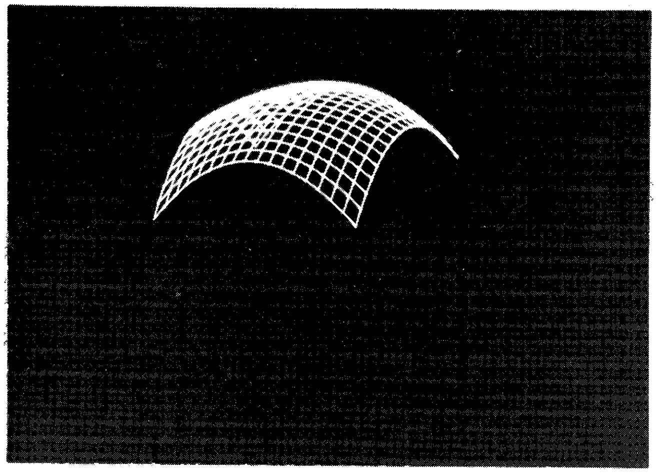


Figure 12—Example for the nine point method of defining a surface patch

The nine point method works better the more information about the objects one has (points of inflexion, boundary, etc.). Boundaries are approximated in an optimum way if they are parabolic. Any time a break point or a point of inflexion occurs, a new surface patch has to be defined. Hence, the nine point method leads to a finer partitioning than COONS' method. Another disadvantage is that it does not guarantee

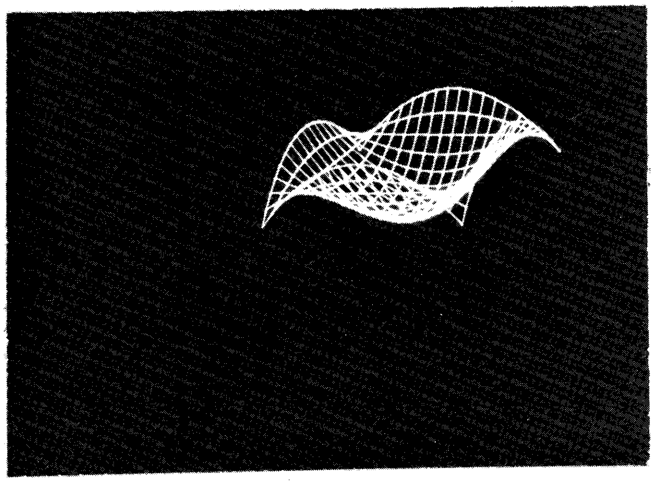


Figure 13—Another example for the nine point method

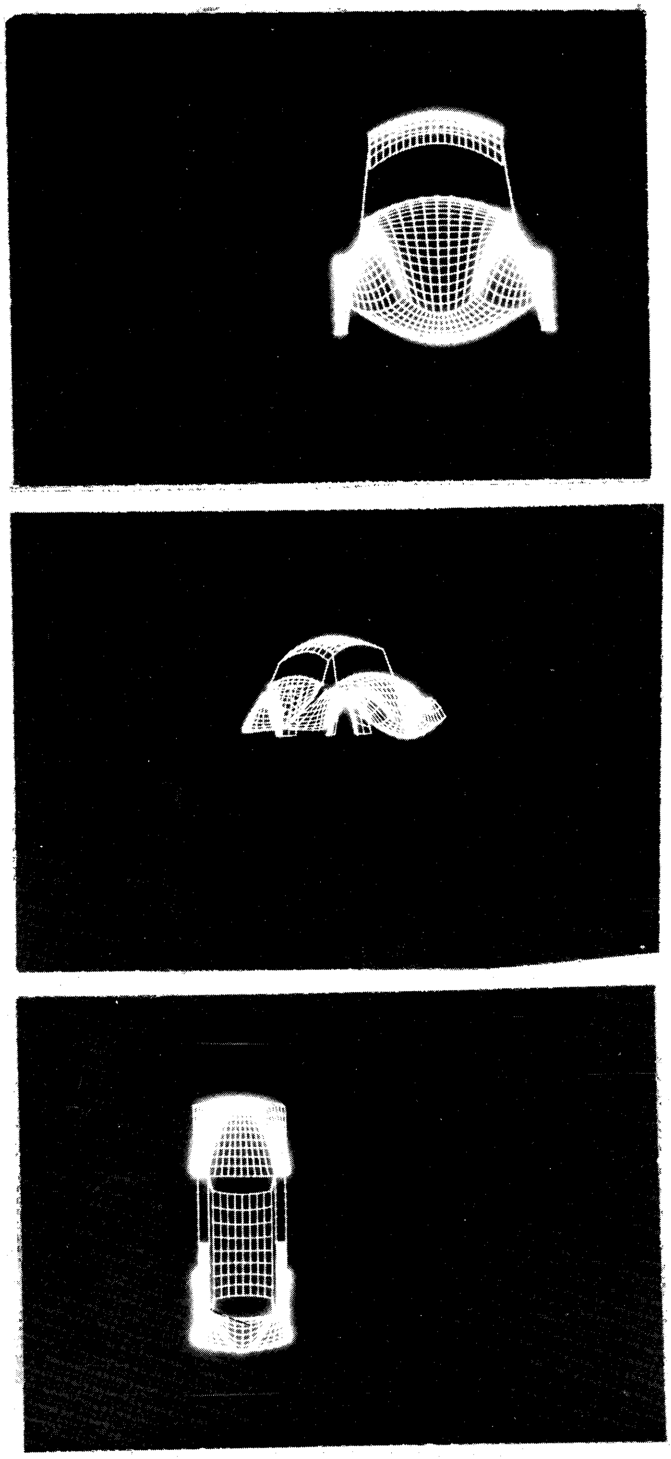


Figure 14—Automobile as an object generated by aid of surface patches

a continuous transition between surface patches. A big advantage, on the other hand, is given by the fact that no blending functions have to be defined and no spatial vectors have to be specified, both requirements expecting very much from the user with respect to his power of imagination.

Additionally, the nine point method requires much less time for preparation and execution of a problem. It is particularly suited for plotters or displays that work incrementally. The man-machine-interaction is excellent. Figure 14 shows as an example the body of a well-known automobile.

COORDINATE TRANSFORMS

PRADIS uses homogeneous coordinates. A coordinate transform is performed by multiplying the matrix of homogeneous point coordinates by the transform matrix TM—as indicated in Figure 15. The transform parameters have to be specified by the user.

VISIBILITY CRITERIA

Visibility criteria enable the program to detect (and erase) hidden lines. For a satisfactory display of three-

$$\underset{\substack{4\times 4\text{-}\\\text{Trans-}\\\text{forma-}\\\text{tion}\\\text{matrix}}}{\begin{bmatrix} \\ \text{TM} \\ \\ \end{bmatrix}} = \underset{\text{Rotation}}{\begin{bmatrix} & & & 0 \\ & \text{RM} & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}} \cdot \underset{\text{Scale}}{\begin{bmatrix} Sx & & & \\ & Sy & & \\ & & Sz & \\ & & & 1 \end{bmatrix}}$$

$$\cdot \underset{\text{Translation}}{\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ X_T & Y_T & Z_T & 1 \end{bmatrix}} \cdot \underset{\text{Perspective}}{\begin{bmatrix} 1 & & & 0 \\ & 1 & & 0 \\ & & 0 & -1/c_z \\ & & & 1 \end{bmatrix}}$$

$$[x \quad y \quad z \quad 1]\cdot[\text{TM}]=[x' \quad y' \quad 0 \quad p] \qquad p=1-z'/c_z$$

c_z . . . Centrum of projection

Picture coordinates $X=x'/p$

$Y=y'/p$

Figure 15—The 4×4—Transformation matrix

dimensional objects, the hidden line suppression is mandatory. We distinguish between three types of criteria: point tests, surface tests, and combined point/surface tests. In the case of surface tests the primitive which is tested is a surface element as the name indicates. The surface test indicates only, whether or not a line is visible as an entity; i.e., a possible mutual hiding of two surface elements cannot be taken into account. Hence, it is not applicable to concave structures.

Points test partition a line into small segments that are only considered to be visible if none of the arbitrarily selected points on this segment is hidden by a surface element. The point test procedure is generally applicable, however, it requires a prohibitive amount of execution time. Combined point/surface tests try to combine the advantages and avoid the disadvantages of point tests and surface tests.

In PRADIS, we use some new schemes which we developed.[12] The implementation of our hidden line detection algorithms are called FLAVER, FLAKA, and FLAVI. The gist of these three algorithms shall be summarized in the following.

(1) FLAVER—FLAVER uses a hidden line algorithm that is applicable to convex structures. The procedure consists of three main steps. In the first step, the angle between the line-of-sight and the normal line on the surface is calculated resulting in a criterium for totally hidden surfaces. In the second step, a priority is assigned to each one of the remaining surfaces. Thus, the sequence is defined in which the mutual hiding will be determined. This is done in the third step by investigating whether or not the terminating points of a line fall into a surface with higher priority than that of the surface the line belongs to. If only one of the terminating points is hidden, the point of intersection between this line and the surface boundary is determined. This point terminates the visible part of the line. The point is that only surfaces with higher priority than that of the surface the respective line belongs to are to be taken into account, thus saving considerably computation time.[12,13]

(2) FLAKA—FLAKA works on concave structures; i.e., it works in the more general case where the polyhedrons may have cavities and/or a group of disconnected polyhedrons occurs. In this case, the angle between the line-of-sight and the normal line, unfortunately, does not provide a simple criterium for determining whether a surface is visible. This difficulty is overcome by dividing in a preliminary step all the surfaces of a structure into triangles. On these triangles the last two steps of FLAVER can be applied. The partitioning of all surfaces of the structure results in additional lines (if a rectangle is partitioned into two triangles, an additional line is generated which is the rectangle diagonal). Special care has to be taken to suppress these additional lines (therefore, the triangles are of course not visible).

(3) FLAVI—The intrinsic hidden line detection algorithm of FLAVI takes care of objects that are defined by analytical expressions—be these expressions defined by the user in form of equations or generated by the program according to COONS' theory. The procedure works as follows: First of all a Cartesian coordinate grid of 11 by 11 lines is superposed. If the visibility of the point of intersection between $2(\Delta u - \Delta v)$-segments has to be determined, only the surface patches have to be considered that have no empty intersection with the respective grid area. Subsequently, surface patches to be taken into account are partitioned into triangles, and the z-coordinates of these triangles are calculated at values of x and y given by the coordinates of the intersection point. If a triangle has a z-coordinate which is greater than that of the intersection point, it hides that point. If no such triangle exists, the point of intersection is visible. All points of intersection of all lines $u=$constant and $v=$ constant have to be tested in that way. If one of two connected points is visible and the other one hidden, more points located on the connecting contour have to be examined.

ACKNOWLEDGMENTS

The authors are gratefully indebted to Mssrs. Grosskopf, Schulz, Castro, Mahramzadeh, Hunger, Burmeister, Eckert, and Kniepen who contributed by their thesis work under our supervision to the successful implementation of the PRADIS system.

REFERENCES

1 H J BEISTER
Zur Technik der Segmentierung von Programmen bei Rechenanlagen mit kleinem
Speicher edv 7/69

2 T JOHNSON
SKETCHPAD III—A computer program for drawing in three dimensions
Proc SJCC 1963

3 D EWING D TEDD
A display technique for manipulating engineering drawing in 3 dimensions
CG 70—London

4 R J HUBBOLD
TDD—A 3 D drawing program
CG 70—London

5 J ENCARNACAO J HUNGER
Das interaktive Zeichnen und Entwerfen in drei Dimensionen
Elektronische Rechenanlagen 1 February 1971

6 J CASTRO H MAHRAMZADEH
Beschreibung, Untersuchung und Erprobung der Zerlegung beliebiger Koerper mit konkaver Struktur in konvexen Ebenen
Studienarbeit TU Berlin 1969

7 J ENCARNACAO
Untersuchungen zum Problem der raumlichen Darstellungen auf ebenen Bildschirmen
Doktorarbeit TU Berlin 1970

8 R ECKERT
Untersuchung und Verallgemeinerung von FLAVI Studienarbeit
TU Berlin 1970

9 J ENCARNACAO
A solution to the problem of reconstructing objects from three views and displaying them using a new hidden-line algorithm
DATAFAI 71 March 1971 Nottingham

10 S A COONS
Surfaces for computer aided design of space forms
MAC—TR—41 Clearinghouse for Federal Scientific and Technical Information Springfield Virginia 1967

11 G SCHULZ
Flächeneingabe durch Eckpunkte und Tangentenvektoren und durch neun spezielle Punkte
Diplomarbeit TU Berlin 1970

12 J ENCARNACAO
A survey of and new solutions for the hidden-line problem
CG-Symposium Delft 1970

13 G GROSSKOPF
Beschreibung, Untersuchung und Erprobung der Visibilitaetsverfahren FLAVER und FLAKA Diplomarbeit
TU Berlin 1970

MARS—Missouri Automated Radiology System

by J. L. LEHR, G. S. LODWICK, L. J. GARROTTO, D. J. MANSON and B. F. NICHOLSON

University of Missouri
Columbia, Missouri

The primary role of the radiologist is to examine patients, usually with the help of ionizing radiation, in order to provide information of use in patient care. The radiologist functions as a consultant that is, patients are referred to him by many other physicians, and he delivers information obtained from his special methods of examination back to each patient's referring physician. As a result the radiologist deals with greater numbers of patients than most physicians. Also, he needs to move a great deal of data quickly and accurately.

Radiology is a relatively young field of medicine, and the variety and complexity of examination techniques in the radiologist's aramentarium continue to multiply. Because of this and because he examines more patients each year, the annual increase in demand upon a radiologist's time has been estimated from 15 to 25 percent. Fortunately, however, radiologists are used to living with complicated electronic gear, and the specialty as a whole has shown considerable interest in employing data processing techniques to streamline patient care delivery.

Our own efforts began in 1965 as project RADIATE (Radiologic Diagnoses Instantaneously Accessed and Transmitted Electronically), continued as ODARS (On-line Diagnostic and Reporting System), and have finally been renamed MARS (Missouri Automated Radiology System). It was apparent initially that straightforward computer programs could carry out a variety of business functions such as patient billing or inventory control. However, the most important information flowing through a radiology department is the medical information, i.e., what the radiologist has to say about the patients he examined. The processing of this basic information was the problem we tackled.

Traditionally the radiologist has dictated his consultation after reviewing a patient's films. This dictation is then transcribed, proofread, and finally returned to the referring physician. The problem of capturing this data for a computer can be solved in a variety of ways ranging from keypunching the reports to utilizing typewriters which produce magnetic tape along with typed copy. Such approaches do provide the computer with data to process. Indeed, they provide a large amount of natural language data which requires very sophisticated processing to interpret. Although processing of dictated reports has the superficial advantage of not interfering with traditional methods, it tends to complicate the flow of information rather than facilitating it.

We have chosen a more radical approach for MARS. Instead of dictating his report, the radiologist interacts with the computer via an on-line terminal. He constructs his report by retrieving terms from several large lists, each containing a particular category of term, and stored in direct access files. The retrieval technique is a modified keyword in context approach. The radiologist types in one letter denoting the particular type of term followed by the first four letters of any word in the term. The computer displays a list of all terms associated with this keyword, and the radiologist selects one of these. Three categories of terms are available; examination types, anatomic sites, and diagnoses.

In addition to incorporating phrases retrieved by keyword, the radiologist has additional options in preparing his report. He may select any of a list of commonly employed phrases such as "there has been essentially no change in findings since the previous examination." This particular phrase being the fourth in the list, the radiologist may simply type "P4" to add it to the report. Or, if he does not remember the number, he may simply type "P" to request display of the entire list from which he then makes his selection. Another option available is the incorporation of a pre-coded sequence of statements which are retrieved by typing a three letter abbreviation, or mnemonic. For example, if the radiologist types "CAD," the computer will incorporate into the report a series of statements describing a geriatric chest with calcification in the aortic arch, tortuosity of the descending aorta, and degenerative changes in the thoracic spine. These pre-coded statements frequently make it possible to complete an entire report by typing only a few letters. The approach is obviously quite similar to the use of "canned" reports generated by semi-automatic typewriters. However, the computer system does have the advantage that the radiologist reviews the report on his terminal immediately and, if he wishes, he may easily edit it. For example, if the changes in the spine are unusually marked, he might add an appropriate modifier to the report. Finally, if the radiologist is unable to express himself fully by the options provided, he may type as many additional remarks as he feels are needed.

This method was aimed at solving the basic problem of capturing the radiologist's report in a form which the computer could comprehend. The technique met with criticism for two reasons. First, it altered the traditional form of the radiological consultation. Instead of paragraphs of prose, the referring physician would receive a somewhat terse list of observations and impressions. Second, it required that the radiologist operate a keyboard. Added to the very real problem which would arise if the radiologist were slowed down by using a terminal there is a largely emotional problem which stems from the idea that it is somehow "sissy" for an M.D. to type.

The only way we could see to answer these objections was to put the System to work. Our immediate goal was not to automate all the functions of the department, but to test the hypothesis that the medical information flow could be handled by an on-line system as described. Accordingly we wrote, tested, and rewrote programs. Finally, on April 1, 1970, we locked up the radiologists' dictaphones and began using MARS to process all consultation in our department. Clinical use required not only that the radiologist be able to interact with the computer to define individual reports but also that there be a system for handling the flow of patients through the department. The system we have used can best be described by following the progress of a typical consultation through the department.

A patient presents for examination in the department at a reception area accompanied by a request form filled out and signed by the referring physician. The receptionist informs the computer of the patient's arrival by entering the patient's unit number via a terminal. The computer searches the hospital master disk file for this number and responds by displaying information found there, including the patient's name, date of birth, sex, race and hospital ward or outpatient clinic code. If the information is not on file (as, for example, in the case of a patient not previously seen at the hospital who requires emergency care), it is entered by the receptionist manually. The receptionist then types in the clinical history provided by the referring physician, along with his name. Finally, the examination requested is entered, and the computer stores all of this information in a temporary holding file. A three digit number indicating the location of the record within this file is displayed to the receptionist who writes it on the request form.

The patient is examined and the roentgenograms, along with the patient's previous X-rays and reports and the request form are brought to a radiologist at one of several terminals for interpretation. The radiologist begins by entering the three digit address of the information entered by the receptionist. The computer retrieves the preprocessed patient identification data and displays it on the screen. The radiologist corrects any errors in this data and then completes the report by the method outlined above. Finally the radiologist proofreads the entire report, makes any corrections and then signals the computer that the report is complete. The computer immediately types out a copy of the report on an output terminal located on the patient's ward and, simultaneously, three copies are produced in the department of radiology. One of these copies is affixed to the jacket containing the patient's radiographs; the second is mailed to the referring physician for his convenience; and the third is attached to the request form, presented to the radiologist for signature and returned to the patient's medical record.

All the reports transmitted during the day are kept on a random access disk file and any of them can be reviewed instantly by entering any patient's unit number on a terminal. At night, each report transmitted is permanently stored in a numerically coded form on a large direct access file. From this file any report for any patient can be retrieved for review within a matter of seconds. Also categorical searches can be performed on-line.

In our two years of experience with this system we have learned many things. Primarily we have shown that it is possible to operate a department with an on-line information system. In evaluating MARS we did measure some parameters such as the time spent by patients in the department and the time a radiologist spends reporting a case. We found these to be essentially unchanged from data accumulated before the System went into operation. MARS did, however make a great reduction in the delay between arrival of a patient in the department for examination and delivery of a written consultation. Before MARS this delay averaged 23 hours with about 75 percent of all reports being sent out on the day following the examination. With MARS this delay has been reduced to an average of 10 hours, and 75 percent of all reports are transmitted on the day of the examination. We regard this as a significant improvement in patient care.

Our experience operating MARS has not been uniformly pleasant, however. We have been plagued by unreliable computer services. We have been operating on an IBM 360-50 under O.S. using the Baylor Message Handler for terminal I/O. The system is a busy one with two million bytes of "slow" core, multiple tapes, disks, and a data cell. In addition to a couple of batch partitions, the system handles a variety of on-line applications for other departments in the Medical Center, as well as a good deal of developmental work. With all of this, the time during which MARS has not been available to radiology has averaged slightly over 10 percent of our working day. For the most part, failures are due to system software. The system simply dissolves. Generally it can be restarted in from fifteen to twenty minutes providing our radiologists with an unscheduled coffee break. More serious failures of hardware or software have not been uncommon, however, and when the system is down for an hour or longer we revert to dictation of reports.

We should emphasize that our computer center has a staff of good systems programmers and is headed by an able Director. However, we see certain disadvantages in the very concept of a large central computer. If a radiologist is going to build his practice around an information system, then he will be responsible for the effect that system has on patient care. This responsibility is very difficult to assume unless it is coupled with clearly defined administrative control over the operation of that system. It is difficult to see how this is possible if the radiologist is only one user of a large time sharing computer.

Our experience with MARS on the 360 has convinced us that the concept of on-line management of information flow is valid for a department of radiology. We have been reluctant, however, to fully exploit the possibilities for a more completely automated department on a computer system with the disadvantages outlined above. Finally, we have become quite interested in building a system which can be exported easily to other departments. For these reasons we are in the process of implementing a stand-alone version of MARS on a smaller computer, the PDP-15, using MUMPS, the Massachusetts General Hospital Multi-Programming System.

The basic reporting function of MARS on the PDP-15 is still quite similar to that discussed previously and consists of retrieving from standardized tables those terms to be incorporated into the reports. We have been able to make the retrieval of terms a bit more sophisticated, however. After examining the frequency with which terms were used in some 50,000 reports we found that a relatively small number of terms were used quite frequently. These terms have been assigned unique three-letter mnemonics so that they can be retrieved without bothering to look at other terms which would be retrieved under the keyword search. In addition, the keywords are no longer restricted to four letters. The modifying terms have been expanded, separated into exam, site, and diagnosis categories, and assigned mnemonic codes for retrieval. Also, the definition of "canned" reports has been left to the discretion of the individual radiologist, so that these can be tailored to suit his needs. Finally, the program has been altered so that the radiologist can specify multiple terms with one input string. This frequently makes it possible for him to define an entire report with one turn-around at his terminal.

In addition to these relatively minor changes at the radiologist-computer interface, we are expanding MARS to automate several other departmental functions. For the most part these are the fairly straightforward business-like applications which we ignored previously, and they come as rather natural spin-offs from the medical information system. These applications are, however, extremely important from a cost-effectiveness viewpoint. One application is patient billing. Since the MARS report contains the name of the patient, the examination performed, and the consulting radiologist's name all the information needed to specify the charge is available to the System. It is only necessary to associate with each examination in the table the appropriate fee. The System displays these charges to the radiologist who verifies and/or corrects them just prior to transmitting his report.

In many departments patient billing is handled by another agency, usually a hospital. Under these circumstances the department's responsibility ends at correctly specifying the charge for each patient. If billing is done by hand, a daily listing of charge should suffice. At our hospital, patient billing is done by computer, so that we will provide the charges on magnetic tape. We do expect to develop a more complete billing and accounting system for those departments which elect to handle this function. Our basic concept is to keep computerized ledger sheets on direct access files. For active accounts, this record will contain both itemized charges and a record of payments made. Inactive accounts will be transferred to tape and/or paper files. The goal is to provide a billing system which is responsive to human direction and can make any patient's bill accessible for review and, if necessary, correction on-line.

Another type of information which MARS can provide automatically is a statistical analysis of the department's work load. The rapid growth of services delivered by radiologists means that additional equipment and personnel must be acquired almost continuously. In deciding just what piece of equipment should be purchased next it is obviously valuable to know what types of examinations are being performed most often. We assign a statistical code to each entry in the exam table which defines for the System how we want our utilization counts broken down. When a report is transmitted the appropriate counts are incremented. These figures are stored by the day during any month. At the end of the month totals are printed out, and at the end of the year a monthly breakdown is produced.

Another operation in the department which is amenable to automation is management of the X-ray files. Currently roentgenographic images are stored on large films. All films for a patient are usually stored together in one or more large jackets which in turn are filed according to patient number. Films are pulled from the files either for use within the department or to be loaned to referring physicians for use outside the department, for use in conferences, clinics or in the operating suite. Since the X-rays themselves form a part of the medical record, the radiologist being legally responsible for them, it is obviously essential to have a system for keeping track of who has checked out what films. It is also necessary at times, to send out "overdue" notices. Current films are also pulled for review by referring physicians within the department viewing area. Since these films do not leave the department, it is not necessary to check them out. However, it is not uncommon for a doctor to request films which have been checked out. Under any of these circumstances a harried search through the department, and through multiple scraps of paper ensues. Some tracking of the films within the department is accomplished by logging the patient in and reporting the case. It would be possible to create additional check points to further ameliorate this problem.

Finally, any time a patient is examined in radiology, all of the patient's previous X-rays are pulled for review so that the radiologist may detect for changes in the area examined, or so that he may correlate the findings from examinations of other areas. In order to reduce the delay caused by pulling the old films, MARS transmits a request to the file area as soon as the patient is logged in at the reception area.

One of the ways in which radiologists attempt to meet increasing patient loads is by keeping existing facilities busy through scheduling of examinations. In our department only those examinations which require the participation of a radiologist have been scheduled. In particular, nearly all patients have been simply sent from the clinics to the department without any prior warning. To alleviate this problem we have developed an on-line appointment book. A referring physician, or his secretary, may call the radiology reception area and request that a given exam be scheduled for his patient at a particular time. The System opens the appointment book to the appropriate day, and checks to see what has already been scheduled. If the schedule is tight, the System warns the receptionist who can then attempt to get another time which is more convenient for the patient, his physician, and the department. Each morning the schedule is printed out allowing our personnel to plan how they will meet the projected patient load. A longer range goal for this MARS module is to provide for a constant on-line reevaluation of the work load by the System, which might then make specific recommendations such as assigning individual patients to specific examination rooms.

A potentially great advantage of having a radiologist interacting with a computer is that the machine may be able to assist them interpret the films. There are several possible techniques for this. Perhaps the simplest of these is retrieval of useful medical data. We have a growing file of radiological information arranged hierarchically by organ system, and subdivided by diseases or roentgen findings. This film may be examined on-line by the radiologist while he is reporting. A slightly more complicated method of assisting the radiologist is to provide him with a logical tree for the analysis of certain diagnostic problems. The radiologist is asked a series of questions about the findings on the films and depending upon his answer to each question another appropriate question is asked until he reaches a leaf on the tree. Typically this will be a list of diagnostic possibilities. Another technique which may be employed is that of Bayesian analysis. The radiologist is asked to specify whether certain findings are present on the films, and the System then uses estimates of the frequencies with which these findings occur in various diseases to estimate probabilities for the diagnoses. Such programs are available for the diagnosis of pulmonary nodules, heart disease, and thyroid dysfunctions. The tree branching and Bayes techniques have been combined into a program for the analysis of solitary bone tumors. Many investigators are becoming interested in collecting the type of data required to attack diagnostic problems by means of these techniques.

There is great pressure on the medical profession to produce more patient care with no loss in quality and without further increasing the cost to society. Computer technology holds the promise of helping us to reach that goal. Our efforts in developing MARS have been to automate the practice of radiology. In doing this we have made changes from the traditional practice of our specialty; we feel that we have improved it.

Certainly there is need for further work. The use of computers for training radiologists and radiological technologists has not been thoroughly investigated. Technological advances on the horizon such as high quality microfilm systems for storage of roentgenograms, or electronic equipment to transmit images will both augment the possibilities for automation, and may well require sophisticated data processing systems to have their full impact. Finally, we are beginning to see some practical possibilities for direct computer analysis of the X-rays themselves. The task of demonstrating the feasibility of automated systems, of developing cost-effective systems, and of educating the medical profession to use them effectively should provide a challenge for many years to come.

BIBLIOGRAPHY

W J WILSON A W TEMPLETON A H TURNER
G S LODWICK
The computer analysis and diagnosis of gastric ulcers
Presented at 13th Association of University Radiologist Meeting Seattle Washington May 14–15 1965 Radiology Vol 85 No 6 pp 1064–1073 December 1965

G S LODWICK P L REICHERTZ
Computer assisted diagnosis of tumors and tumor-like lesions of bone
The Limited Bayes' Concept Proceedings of Symposium Osseum London April 1968

A W TEMPLETON G S LODWICK S D SIDES
J L LEHR
RADIATE: A project for the synthesis, storage and retrieval of radiologic consultations
Digest of the 7th International Conference on Medical and Biological Engineering 1967 Stockholm, Sweden

A W TEMPLETON P L REICHERTZ
E PAQUET J L LEHR G S LODWICK
F I SCOTT
RADIATE-UPDATED and redesigned for multiple cathode-ray tube terminals
Radiology Vol 92 No 1 pp 30–36 January 1969

P L REICHERTZ A W TEMPLETON J L LEHR
E PAQUET F B BIRZNIEKS
Design and implementation of ODARS, an on-line diagnosis and reporting system
Presented by Dr Peter Reichertz at the 12th International Congress of Radiology Tokyo Japan October 1969

J L LEHR R W PARKEY L J GARROTTO
C A HARLOW G S LODWICK
computer algorithms for the detection of brain scintigram abnormalities
RADIOLOGY November 1970

"Sailing"—An example of computer animation and iconic communication

by STEPHEN M. ZWARG

National Security Agency
Ft. George Meade, Maryland

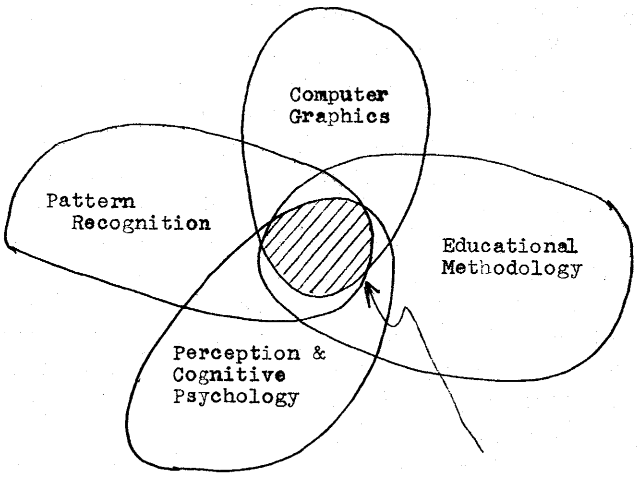
INTRODUCTION

The use of the visual process for the perception and assimilation of ideas is well recognized by educators, psychologists, as well as computer scientists. The popularity of movies and television and the concomitant growth in visual literacy of the viewers of these media are two factors exploited by those who wish to communicate ideas using these media. The world which comes across on the movie or TV screen is the camera's eye view of the visual world (as opposed to our visual field, that which our eyesight and brain perceive of the visual world, or "reality," see Gibson[6]); and as such it is an image of reality. This can include images of beautiful scenery, complex human interactions, or of a printed page illustrating a lecture. Any ideas which are communicated by these devices, while relying upon imagery, tend to inform the viewer by creating the illusion that he is there, so how and why would he act, or they inform him by straightforward strings from some generally accepted language (as in the case of a screen full of text). Bruner, a cognitive theorist, feels that information acquisition proceeds through three stages:[5]

1. Enactive—using personal action
2. Iconic—perceptual organization and integration of images
3. Symbolic—using a language or accepted set of representations

As a child develops he progresses through these three stages. Most of the things we learn in school or from books are by symbolic means. Imagery can back up symbolic and enactive paths of learning with varying degrees of success.

Iconic communication itself, using pictures, figures, or visual images in combination with motion in the cases of movies, cartoons, and TV, is not so well understood as an idea transporter. With the fusion of the abstract symbol manipulative, computational power of the computer and the concrete world of visual imagery (in an animated movie in this case) it was hoped to study the problem of idea expression and perception through images. The motivation for producing a computer animated film was simple—it was a course requirement for a graduate seminar led by Prof. William Huggins at Johns Hopkins University. Titled Iconic Communications, the intent of the course was to produce a movie, and thereby develop and learn the computer related techniques, as well as develop the potential of iconic modes of communication by research in the related areas of: computer graphic systems and devices and data structures; machine perception, pattern recognition, and syntax of pictures; memory, perception, and cognitive psychology; education, geometry, and visual syntax.



Iconic Communications

This sort of interdisciplinary study was interesting as well as very fruitful for those of us in technical fields, such as electrical engineering or computer science, who were looking for new applications or who only vaguely understood present applications of advanced computer graphic technology.

OTHER COMPUTER ANIMATION SYSTEMS

The use of computers for the production of animated movies is not new. Several on-line and off-line methods exist with varying degrees of interactive capability. Baecker[3,4] at M.I.T. produced the GENESYS system, making use of the TX-2 computer and graphics system, which permitted the animator/operator to specify on the graphic screen (1) images to be moved about as well as (2) schedules by which they were to be moved. All of this was done on-line and little or no programming was necessary by the animator. The images used were anything that the user could draw, alleviating him from tediously specifying coordinate input. The feedback of results was immediate. One result of taking programming away from the user was that the computer spent its time keeping track of the animator's on-line doodlings instead of offering computational or problem solving power directly toward what to do with these doodlings. The specification of pictures and their driving functions was very good for simple and fast cases but not really suitable for an extended animated effort. Spending a short amount of time to produce a short sequence tended to make the effect of the image short-lived.

Others, including Anderson[1,2] and Knowlton[8,9] spent their time developing elaborate list processing languages and stacking data structures (as in CAMP and BEFLIX). These specialized languages required much understanding on the part of the animator but his facility for taking advantage of the computer's computational power rose correspondingly. The usual technique was to write the program in the specialized movie language, run it on a computer graphics terminal, film it, and then postprocess the film to add color or special effects. Some strikingly beautiful films were produced by Knowlton at Bell Labs and are available on loan from there. Not much attention was paid in these efforts to what images were used or what ideas should be communicated. Visual effects were sufficient to prove the technology.

More recently, at the University of Pennsylvania's Moore School, Talbot, et al.,[9] have attempted to fuse the on-line interactive approach to movie specification with the remote computational power of a large computer for the final movie production. They also developed a specialized hierarchical language (with a formal Backus Naur syntax specification) for the animator to specify his pictures, motion, scenes, and movie segments. The on-line activity consisted of drawing on the DEC-338 scope, working in one of the picture definition modes mentioned above, and then reviewing the rough construction of the movie immediately. When the animator was satisfied with his rough draft he entered transmission mode and the outline of the movie was sent to a remote IBM 360/65. There the draft of the movie was translated into the final movie generation commands to drive a SC-4020 microfilm plotter. The main point was to be able to review movie production at any stage as well as to specify everything on-line. Data entry was again a problem, figure manipulation awkward, and no concentration was made on types of images, what their effects and relationships should be and how to get ideas across. The preoccupation with hardware and syntax specification has obfuscated idea communication.

THE MOGUL SYSTEM

Our animation system at Johns Hopkins possesses all the so-called drawbacks. It is not on-line, no expensive graphics equipment is available, only a second generation IBM 7094 computer. Turn around for the finished movie is slow since no SC-4020 plotter is owned either. Tapes must be sent someplace that has one, delaying things by several weeks. Short term feedback of a coarse nature is available from the computer's line printer, and outputs can be had within an hour or so, typical of a batch operation. The draw package used by animators, entitled MOGUL—Movie Oriented Graphical Utility Language[7] and developed by Prof. Huggins, Tom Hoffman, and Jerry Yochelson, is a FORTRAN based system and requires that the user understand that widely known language. As such, though, the user has at his command all the computational routines he can program. Data input for animator specified images in two or three dimensions is as easy as FORTRAN allows. The time invested in learning how to program and working out movie scenes on graph paper is well invested, however, and may even overcome all the previously mentioned handicaps, or at least show that lack of equipment is not always a handicap. Those of us with programming experience enjoyed the ability to work with an animation system in a familiar language. The preciseness of a computer language (but not the restriction of a special purpose language) and the slight delay in feedback of results made us be more sure of what we wanted to see, and eliminated the draw it and play it right back looseness of on-line systems. We found that for at least 75 percent of the movie's preparation the immediate feedback is not necessary if we had a good idea (through time spent on a storyboard and basic visual techniques) of how the movie should go. Admittedly the final polishing up would have been easier with immediate checks, but costs prohibited this.

THE IDEA OF THE SAILING MOVIE

Since all of my spare time during the summer is spent on the water, when asked to make a film about an idea, the one which naturally occurred to me was that of a sailboat making its way around a race course, thereby demonstrating basic sailboat aerodynamics. The first step in my animation effort was to lay out a storyboard. This is the sequence of key scenes in the movie along with a brief explanation of what is happening. Briefly the storyboard of the "Sailing" movie goes like this (see Figures 4-9, also an intermediate version 16mm film of "Sailing" and several other students' works is available).[11] An outline of a boat with mast and sail appears, centered on the screen. The boat representation is similar to those found in sailing text and rulebooks, reinforcing its familiarity. The boat then shrinks as a race course appears. The course consists of three flag buoys and wind arrow, pointing from left to right. The boat shrinks to normal size and begins sailing around the course. The boat tacks up the first leg, illustrating the relative position of boat and sail necessary for movement against the wind. The boat reaches down the second leg, jibes, and reaches down the third leg, with the sail farther out moving faster than on the first leg. Midway up the first leg again the boat grows, buoys disappear but wind arrow remains. The next sequence attempts to demonstrate the aerodynamic principles of a sail in terms of force vectors and parallelograms of forces. Three arrows grow out of the center of the sail representing forces acting on the sail. One is parallel to the wind arrow, the sum of forces developed by the wind; the other two are perpendicular and parallel to the surface of the sail, the components. The component perpendicular to the surface of the sail in turn has components parallel and perpendicular to the axis of the boat. The first of these will drive the boat forward while the latter will drive it sideways. With a keel or centerboard the sideways force is largely resisted. The forward driving force acting on the boat then results. Arrows in the forward and sideways directions are drawn. Two force rectangles are now on the screen. They grow and shrink simultaneously with the wind arrow to illustrate that forces developed are proportional to wind velocity. The various component arrows grow and shrink and change direction as the sail is overtrimmed and overeased, illustrating the optimum position of trim—that point at which the forward driving arrow is longest.

Arrows disappear. Boat shrinks back to normal size and continues on its second lap around the now appearing racecourse. After reaching leg three it slowly shrinks to nothing and heads toward the edge of its world.

DEVELOPMENT OF GENERAL ANIMATION TECHNIQUES

One of the goals of the movie experience was to develop some general rules, heuristics, or theories upon which future work could be based. At least we could find out some things not to do. One of the greatest difficulties concerned the use of symbols or images to express ideas. Which are "correct," "good," or "effective," etc? What is the criterion (or grammar) for judging visual images? How does one even make graphic an idea without moving from the iconic to the symbolic level of communication? The medium is hard line black and white to start with. Using any sort of standardized symbols or representations immediately brings on a whole set of preconceptions (or ignorances) and prejudices on the part of the viewer, and if these are not complementary to the idea being shown they can sidetrack the communication process. In the case of the sailboat, I felt that anyone who thought of sailing rulebooks when he saw my sailboat would be thinking of rules and principles, and that was what I was trying to show so it was a useful image.

The use of text or mathematical languages was also discussed. We felt that any reliance upon text should be minimized. Words or letters only detracted from the intent to stay at the iconic level. When used for emphasis, reinforcement, or clarification, however, text has a place. Similarly, mathematical symbols or any other formal symbolic language more often than not gets in the way of the showing, if not also the conceptualizing of visual explanations. Languages are useful here only in the knowledge that they can be called upon sometimes to write down the process being shown. The literacy of the audience with respect to the language being shown (or not shown) must also be considered.

One thing that had to be well worked out in advance and which was a continual cause of surprise after finally viewing the movie was the layout of images or occupancy of the screen. Given a full, two dimensional square or three by four rectangle for a screen we were acutely aware of empty spaces. The full screen has to be used for motion and action for full spatial effect, but by no means should it be filled with clutter. One of the main advantages of the medium is that it is completely

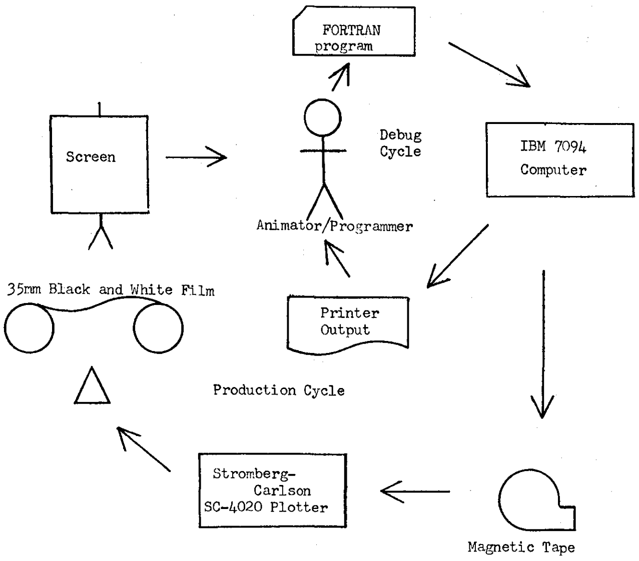


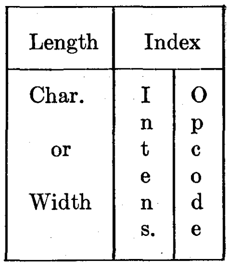
Figure 1—Steps to produce an animated movie

bare bones, stripped of all extraneous detail, and this should be used to best advantage by retaining a simplicity of imagery bordering on paucity. The center of the screen is the key spot for any important activity or attention getting. Split screen or simultaneous occurrences divide attention and effectiveness.

Abstractions and relationships are difficult to portray. The choice of images, their size, relative position, order of appearance and movement all can suggest functions, relationships, and dependencies. There is the difficulty of avoiding coincidental though erroneous suggestions of relations versus the valid suggestions of intended relations. Does the fact that one object precedes another on the screen or that it is larger than another indicate that it is more important? If one event happens before another does it imply cause and effect or time sequence? Questions like this have to be considered for individual cases to be sure that the animator's intentions are being realized. Importance is often indicated by size, positioning at the center, illumination (blinking), and motion. The relationship of cause and effect was never satisfactorily protrayed. In one film the anthropomorphic symbol of a detached hand had to be introduced to cause a switch to be thrown.

The obvious capability of a movie for showing motion cannot be too strongly stressed. Without background sensory cues, however, some sort of frame of reference has to be established to inform the viewer of his or an object's motion. A world of lines, edges, and boundaries is being dealt with, not one of textures and surfaces. Motion of objects within the screen boundaries can mean all sorts of things. The capability of the computer to calculate regular paths also adds to the possibilities. If two objects move simultaneously, or in some synchronized fashion, or with the same pattern an obvious relationship suggestion is being put forward. One problem engendered by the capability of (and the expectation of) motion in the movie is that of bringing images on and off the screen. Should they just appear, pop on, or should they grow or should they come on from the left (or right)? For the illustration of static concepts this is more of a worry than for cases where motion on and off the screen can seem natural.

Timing is the other major tool the animator has to work with, but it is also the most difficult to handle properly. Difficulties in the proper use of the screen area can be resolved using the line printer output, but timing problems never are fully resolved until the completed movie is seen. This is one instance where a preliminary viewing of the movie on a graphics terminal would be useful so that timing could be improved without making a whole other movie. Slowing down the action is one way to illustrate importance, up to the point where things become painfully obvious. Speeding up activity moves the viewer over less important or intermediate material. Judging what fast and slow are is difficult, though, and we really had to see a boring thing happening for an interminable time to understand what too slow was. One rule of thumb we started with to judge how long a sequence should take was to equate the timing with how long a verbal explanation would take. Motion of an object relative to stationary objects or at a faster speed than other objects is one way to command attention. Repetition of action, even with slight variations, is important for facilitation understanding of difficult material. The redundancy of presentation should increase with the complexity of the visual material. It is in the production of repetitive sequences of highly redundant though complex material that the computer comes into its own. Hand animation requires each intermediate frame of a cartoon to be drawn while a computer can painlessly calculate endless



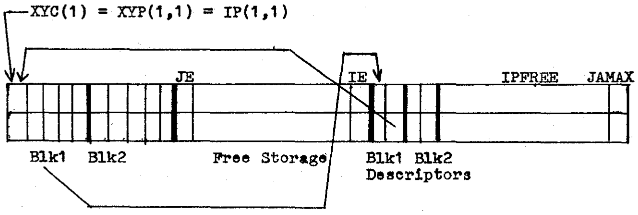
Length	Index	
Char. or Width	Intens.	Opcode

Figure 2—Block descriptor format

numbers of slightly varying scenes. Pauses in activity are very useful as punctuation, giving the viewer time to digest what he has just seen rather than overwhelming him with too much action in too short a time. Fades, holds in action, and jumps to new scenes also serve useful punctuation purposes, each with different effects.

THE MOGUL DRAW PACKAGE

The sequence of steps we followed to produce animated movies is illustrated in Figure 1. The basic tools the animator has are his skills as a FORTRAN programmer, the ability to draw points and lines between them furnished by the MOGUL system, the computer, and the final film medium. The MOGUL system was developed mostly out of our experience in Dr. Huggins' seminar and contains those elements of programming which were always needed for film production, hopefully freeing the animator to deal with image entities and motions rather than coordinate details. It is a structured set of functions built of the compiler level FORTRAN programming language. Those interested in the complete system should refer to Tom Hoffman's paper.[7] Only the salient features of it will be presented.



```
Directory variables:

     IPFREE---pointer to list of free descriptor pairs

     IE    ---highest index of XYC not occupied by block points

     JE    ---lowest index of XYC not occupied by block points

     JAMAX ---maximum dimension of XYC
```

Figure 3—Storage of block information

MOGUL actually consists of three sections. The first and largest is the set of drawing functions, the DRAW package, which enables the animator to create images, manipulate them, and set up operation codes to be interpreted when the images are to be drawn. The second section is the executive portion, MOGI, which keeps track of frame numbers and screen dimensions. The user defines the dimensions of the two dimensional real plane in which he is going to draw things, e.g., a square grid of 1024 points running from 0.0 to 1023.0 in both x and y directions; then MOGI windows any coordinates outside of this area. When output on some sort of device is called for, MOGI normalizes coordinates and then calls on some device driving program. MOGI also administers the mode of movie generation, debugging or production. The third section, the Device Driver section of MOGUL contains the device dependent output code to translate from the normalized coordinates of MOGI and command a printer, microfilm plotter, or scope to produce visual images. The animator's program is actually a subroutine called MOVIE which is called from MAIND, the first program entered. MAIND initializes the system, calls MOVIE, then finalizes the system and returns to the IBSYS Monitor. The user must return himself (via a RETURN statement) to MAIND to terminate his job properly.

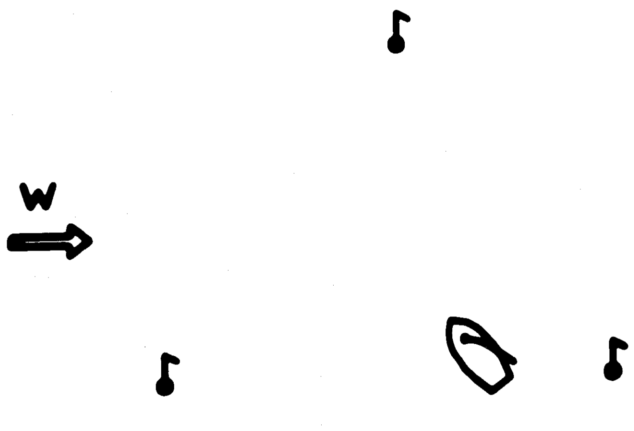


Figure 4—Boat sailing up first leg of racecourse

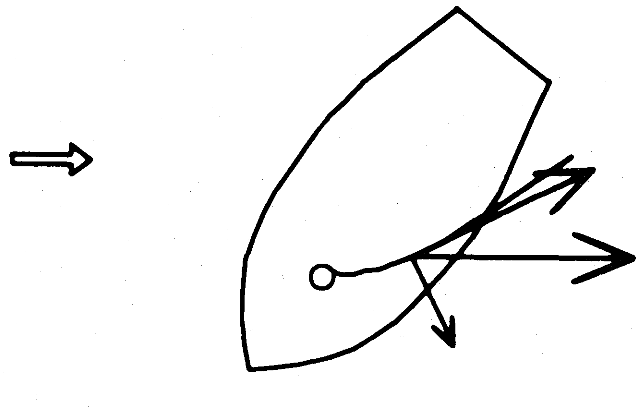


Figure 5—Decomposition of first force vector

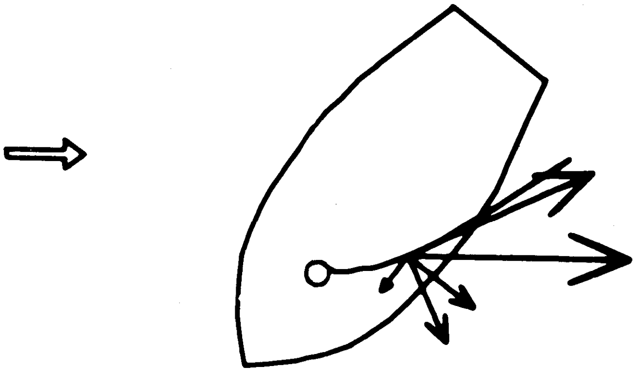


Figure 6—All force vectors depicted

The screen is a partition of the two dimensional real plane with the limits of the partition defined by user specified parameters. The routine is:

CALL SETAX (ZLL, ZUR)

or

CALL SETAX (XLL, YLL, XUR, YUR)

where (XLL, YLL) and (XUR, YUR) are the coordinates of the lower left and upper right corners of the window enclosing the usable screen area. The screen can also be considered a complex plane with points whose coordinates are given by a real (x) part and a complex (y) part. Thus ZLL represents the lower left and ZUR the upper right points just mentioned, and

ZLL = CMPLX(XLL, YLL)

ZUR = CMPLX(XUR, YUR)

Any object which appears on the plane is a collection of x and y coordinate points with lines drawn between them. An object is then a $2 \times N$ real array or block, with the first row being the x coordinates and the second row being the y coordinates, or a $1 \times N$ complex array with the real part equal to the x coordinate and the complex part equal to the y coordinate of a point. Blocks have associated with them names, e.g., Boat, Sail, etc., somewhat indicative of what the coordinate points represent. This name of the block contains a pointer to the location of directory information describing the block. The "value" of the real FORTRAN variable Block is the index of the array XYC where the directory information for Block resides. Included in the directory is the length of the block; its index (a pointer to the location of the block's coordinates in the array XYC); the opcode which specifies whether no lines,

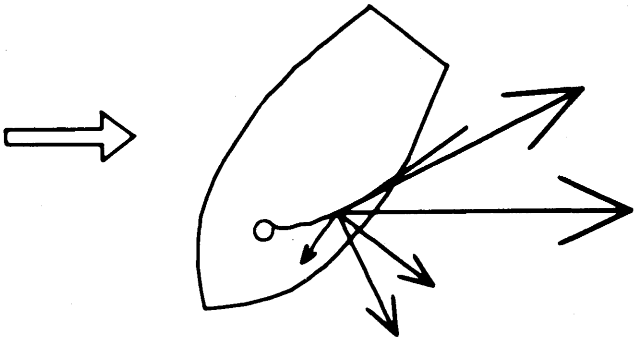


Figure 7—Force vectors and wind arrow enlarged

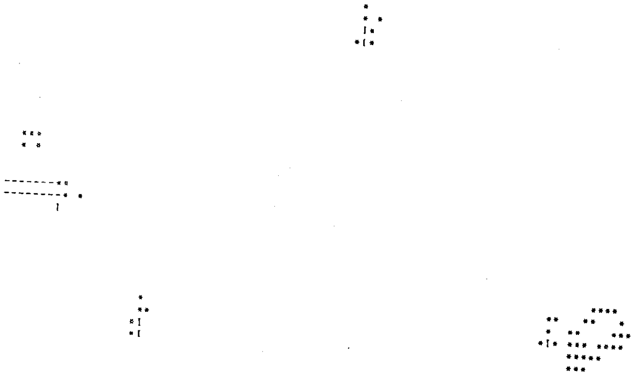


Figure 8—Line printer output showing racecourse

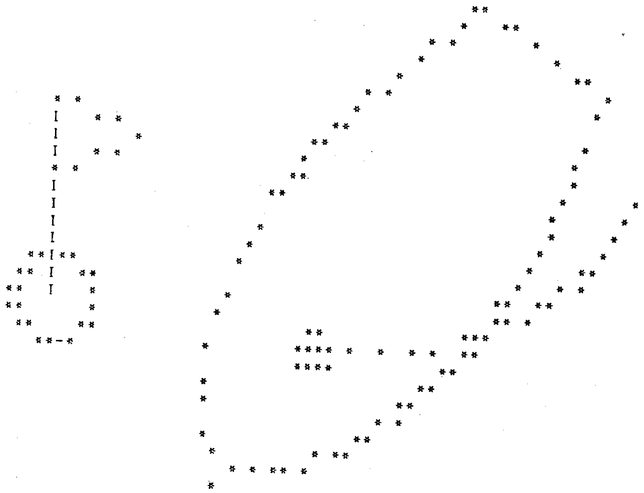


Figure 9—Line printer output, closeup of boat and buoy

disconnected lines, or connected lines should be drawn between points, or where characters should go; the width of lines; and the intensity of plotting. Everything on the screen is specified by a block, facilitating the use of general manipulation routines which always operate on blocks. Only by the opcode of a block is the interpretation of its contents determined.

Storage is considered to be a one dimensional contiguous complex array, and all blocks and descriptors are stored in this array. The array has three equivalent names: XYC, XYP, and IP, depending on whether one likes to work with complex numbers, real numbers, or integers, respectively, as coordinates. In the lower end of XYC go the block contents. In the upper end go the block descriptors. Creations, length adjustments, and deletions of blocks are administered using the index variables of XYC. XYC is declared in a COMMON statement, making it available to any subroutine of the animator.

Figure 2 shows the contents of the directory for a block. Figure 3 depicts the allocation of storage for block information.

Storage for blocks is allocated and all housekeeping of pointers is taken care of by the following functions:

CALL CREATE (N, BLK)

or

BLK=CREATE (N, BLK)

—makes space for a block of size N and sets BLK equal to the pointer to the block directory. The value equals the returned pointer.

BLK=SETUP (N, TYPE)

—creates a block big enough to contain N things of type TYPE, where TYPE can be connected lines (N−1 points needed), disconnected lines (2N points), or characters. Other directory information is set and the value of the function equals the pointer to the block.

CALL COPY (FROM, TO)

or

TO=COPY (FROM, O)

—copies the block FROM into the block TO, adjusting the length of TO if necessary. Directory information is copied also.

Blocks are erased and their space returned to free storage by:

CALL DELETE (BLK1, BLK2, . . . , BLKn)

As can be seen block functions can be effected either by subroutine calls or by function calls which return a value. There are cases in writing a movie program where both are useful, though consistency in calls is to be preferred to avoid errors. Primitive functions are available to set or return the value of block directory information if this is necessary.

Objects to be drawn can be specified in two ways. Either the animator plots his drawing on graph paper and then enters the point coordinates, or he specifies some function to calculate the points. The "Sailing" movie uses both methods. For example the boat and sail were entered as points from a drawing on graph paper, while the mast and buoy were calculated from an equation for the points on a circle. Thus, blocks must be prepared to receive data in several ways. Data can be loaded point by point into a block with the following sequence:

CALL BUILD (BLK)

—initializes the function COORD to begin placing data into BLK beginning at the first column of BLK.

CALL COORD (Z)

or

CALL COORD (X, Y)

—places the complex coordinate Z or the real coordinates X, Y into the next column of BLK.

A whole array of coordinates can be loaded into a block using:

CALL LOAD (I, N, XY, BLK)

or

BLK=LOAD (I, N, XY, BLK)

—loads N coordinates from the complex array XY into BLK beginning at the I column. The second usage returns the pointer to BLK as its value.

Once objects to be displayed have been created and assigned to blocks, the animator must decide what to do with them. Scaling and translation are the two main manipulations he may perform, and these are done by applying arithmetic operations to blocks (which in fact contain complex, real, or integer numbers, depending upon interpretation). Complex variables and complex arithmetic permit handling of two dimensional data with one statement. Scaling is done by a series of complex multiplications of block points by a complex scale factor. Translation is done by adding a complex translation factor to every block point.

CALL SCALE (SFACT, FROM, TO)

or

TO=SCALE (SFACT, FROM, O)

—scales the block FROM by the complex scale factor SFACT and places the results in TO

```
CALL TRANS (TFACT, FROM, TO)
```

or

```
TO=TRANS (TFACT, FROM, O)
```

—translates the block FROM by the complex translation factor TFACT and places the results in TO

```
CALL SCLTRN (SFACT, TFACT, FROM, TO)
```

or

```
TO=SCLTRN (SFACT, TFACT, FROM, O)
```

—scales and translates

With the use of a scale factor defined as follows:

```
SFACT=CMPLX(FACTR, 0.0)
          *CMPLX(COS(ANGLE), SIN(ANGLE))
       =MAG (FACTR)*ROTD(ANGLE)
```

an object will be FACTR times larger and rotated by ANGLE degrees from its former position. MAG and ROTD are defined as above in MOGUL. Neither scaling and translating operations change the data in the block FROM. Scaling of an object occurs about the origin about which the object was defined. If the origin is the origin of the screen coordinates rotations and magnifications will be with respect to this point. A more useful convention is to define all objects with an origin within them so regular rotation and scaling will result. Translation occurs with respect to the object's previous position.

Objects are drawn using the following routines:

```
CALL DRAW (BLK1, BLK2, . . . , BLKn)
```

—causes the output of the n blocks to the Device Driver program where they are drawn according to their directory information.

```
CALL DRAWST(SF1, TF1, BLK1, SF2, TF2,
        BLK2, . . . , SFn, TFn, BLKn)
```

—causes the scaling by SFi, translation by TFi, and drawing of a copy of BLKi. The copy is then deleted leaving BLKi intact. Argument list is one or more triplets.

To roll the film between frames, one logically calls:

```
CALL ROLL
```

—advances film 1 frame.

or

```
CALL ROLL (NFRMS)
```

—advances film NFRMS frames.

ASSEMBLING A MOVIE

Working from his storyboard, the animator knows what objects he wants to display when. Once he has entered or calculated the point coordinates of an object in his program he must specify the dynamics of the object. The usual method for doing this in FORTRAN based MOGUL is with DO loops, where the scale and translation factors change incrementally with the iterations in the loop. This is fine if only one object is being displayed or if all the objects are scaled and translated with the same factors. When different sized objects or different motions are required there must be nested loops or different factors for each. A movie is then a series of DO loops where each loop describes some sequence of frames and contains the following basic elements:

```
    DO 10 I=1, NFRMS
    ——
    SFACT (I)=——
    TFACT (I)=——
    ——
    CALL DRAWST (SFACT(I), TFACT(I),
      BLOCK)
    ——
    CALL ROLL
10  CONTINUE
```

The computer is doing the job of interpolating and specifying all the intermediate frames here, where in conventional animation each frame has to be specified individually. Even commercial animation studios are getting away from this frame by frame drudgery, though, by using over and over certain stock positions, motions, and expressions for characters. The result is somewhat crude and jerky animation. One should really see the old Walt Disney full length animations to appreciate the Rolls Royce versus the Chevrolet approach to animation which is apparent now. Of course no one can pay 750 artists fifty cents an hour anymore either.

DO loops progress at a linear rate and all motions are not necessarily linear. Variable increments in translation factors must then be calculated. Another difficulty arises in the drawing of an object composed of various smaller objects. Other than by nesting subroutine calls or drawing each component separately in its assembled position, the big object cannot be drawn all at once as one large block. There is no hierarchy for nesting blocks composed of other blocks. In the "Sailing" movie the boat consisted of a hull, a mast, and a sail all of which had to move and rotate as an entity. A subroutine was defined which in turn called all these component blocks and then drew them with the specified parameters.

All of the objects which appear in the film are defined as prototypes. Some I drew myself, such as the boat, arrows, and basic sail. Others, such as the wind modified sail and the mast (a circle) result from calculations. The prototype blocks all have an origin within them. For manipulations either the implicit scaled and translated copy produced in DRAWST or an explicit scaled and translated copy of the prototypes is used to retain the original models.

CONCLUSIONS

The final movie which I produced for the seminar was really only an intermediate version, since the MOGUL system was not complete then. However, it illustrates all the points I wished to make, and only a few bugs and the inconvenience of not having all the DRAW functions sets it apart from the latest version. After watching it and watching others' reactions to it I am satisfied that it is getting some of the ideas across I wanted. The film did reveal to me that I must have had an audience in mind which had some sailing knowledge. Many more of the little points were picked up by these people than by those with no sailing background. The latter saw an interesting little boat move about the screen, but without any sympathy for the problems of sailing they missed the basic points of tacking into the wind and correct sail trimming. The strongest point of the film (and possibly, the medium) is its absolute attention getting capability as well as the elimination of all extraneous background material.

There are limitations to this method of communication. Computer animation has many of the strengths of a computer directed system but the artist as animator is farther removed from his final product. Though he is spared the necessity of drawing many slightly varying drawings and though he can calculate, modify, and display phenomena not observable or feasible in the natural world, he cannot express the artistic technique or capability that would be evident if he were holding the pen. Rather than an artist, an artist-programmer-psychologist-computer scientist is required. The attempts mentioned earlier to provide online animation facilities to those uninitiated in the ways of computers only debilitates the capabilities of both parties. Advantage should be taken of the requirement that the animator have some technical knowledge of the system as well as some awareness of the problems involved in iconic communications. Perhaps a simple enough language or computer will be developed some day which will enable an animator to think out and then immediately create his cartoon sequence. At the moment I think this is not true, nor is it necessarily a bad thing either. The preciseness of computers actually aids in formalizing the images to be used in a communications process.

The results for iconic communications I think are just beginning. The visual medium is good, but it takes too long to get thoughts displayed just right. Other than generalized heuristics there is no formal syntax or set of models which can help a potential image communicator. Measurement of the effectiveness of idea transmission, validity of images used, relationships portrayed, all these subjects still pose many questions. Thoughts have been entertained to match up the computer with a TV screen and dump digital information in a video format onto black and white or color monitors. Technically feasible (and marketed) the concept may lower the cost and time involved in producing animations, as well as bring the manifestation of computer generated images closer to home.

REFERENCES

1 S E ANDERSON
A list processing system for effectively storing computer animated pictures
Technical Report TR-67-6 Electrical Engineering Department Syracuse University Syracuse NY 1967

2 S E ANDERSON D D WEINER
A computer animation movie language for educational motion pictures
Proc AFIPS 1968 FJCC Vol 33 Pt 2 AFIPS Press Montvale NJ pp 1317-1320

3 R M BAECKER
Interactive computer-mediated animation
PhD Diss Dept of Electrical Engineering MIT Cambridge Mass March 1969

4 R M BAECKER
Picture driven animation
Proc AFIPS 1969 SJCC Vol 34 AFIPS Press Montvale NJ pp 273–288

5 J S BRUNER
Toward a theory of instruction
Harvard Univ Press Cambridge Mass 1967

6 J J GIBSON
The perception of the visual world
Riverside Press Cambridge Mass 1950

7 T V HOFFMAN
MOGUL: Movie Oriented Graphical Utility Language
Manual for Iconic Communications Seminar Dept of Elec Eng The Johns Hopkins University Baltimore Md Sept 1971

8 K C KNOWLTON
A computer technique for producing animated movies
Proc AFIPS 1964 SJCC Vol 25 Spartan Books NY pp 67-87

9 K C KNOWLTON
Computer produced movies
Science 150 Nov 1965 pp 1116-1120

10 P A TALBOT J W CARR R R COULTER R C HWANG
Animator: An on-line two-dimensional film animation system
Comm ACM 14 4 April 1971 pp 251-259

11 S M ZWARG D KASIK I SMITH
Sailing, make-break switch, and station break
16mm black and white silent film available from author or Iconic Communications Seminar Dept of Elec Eng The Johns Hopkins Univ Baltimore Md

BIBLIOGRAPHY

1 W H HUGGINS D R ENTWISLE
Computer animation for the academic community
Proc AFIPS 1969 SJCC Vol 34 AFIPS Press Montvale NJ pp 623-627

2 W H HUGGINS D R ENTWISLE
Exploratory studies of films for electrical engineering
Report to US Office of Education Sept 1968 Dept of Elec Eng The Johns Hopkins University Baltimore Md

3 W H HUGGINS
Film animation by computer
Mechanical Engineering Feb 1967 p 26

Computer description and recognition of printed Chinese characters

by WILLIAM STALLINGS

Honeywell Information Systems Inc.
Waltham, Massachusetts

INTRODUCTION

An approach to pattern recognition

An increasingly important aspect of computer pattern recognition research is automatic pattern description. Investigators have emphasized that pattern analysis should be basic to any pattern recognition scheme.[4,11] Pattern analysis may be defined as the identification of elements of a structure and the description of the relationship among those elements. Chinese characters, by virtue of their regular structure, seem well suited for pattern recognition based upon pattern analysis.

With this point of view, a scheme for automatic pattern recognition has been developed which includes the following tasks:

(1) Description. A systematic scheme for the description of the pictorial structure of the patterns to be recognized is developed.
(2) Analysis. An algorithm is designed which analyzes the structure of the patterns, producing a representation of the structure conforming to the descriptive scheme.
(3) Encoding. From the structural representation of a pattern, a code is generated which uniquely identifies the pattern.

This method has been applied to the recognition of Chinese characters. A program has been written which analyzes Chinese characters; the program produces a data structure which describes a character in terms of basic picture elements and the relationship among them. A procedure has been developed for generating a numeric code from the structural representation. Recognition is achieved by building up a dictionary matching characters with their codes; the code for any new instance of a character can then be looked up in the dictionary.

The methodology of pattern recognition still lacks any solid theoretical foundation. The author feels that progress in this area requires building up a body of effective pattern recognition techniques. It is hoped that the methods and techniques developed in this project on a specific class of patterns will be applicable to other pattern recognition problems.

Specifically, this project deals with a class of patterns which displays a rich structure developed from a small number of basic elements, all of which are relatively simple. The patterns are characterized by the fact that these elements may be combined in many complex ways. Other classes of patterns which fit this description will hopefully benefit from knowledge gained here.

A Chinese reading machine

Two obstacles have hindered the access of interested non-Chinese groups to the vast body of written Chinese produced each year. The first is the difficulty of the language itself. Chinese is very complex and takes so long to master that few Westerners ever learn it well. And second, of course, is the size of the printed output in Chinese. Manual translation is slow and tedious, and can never be relied on to handle more than a tiny fraction of the material.

To make available to Westerners the culture and technology of one-quarter of the human race, some form of automation must be introduced. A Chinese reading machine, which could scan printed Chinese and produce English output, would provide the most desirable means of improvement. Such a machine is a long way down the road, but individual steps which advance toward that goal are to be encouraged.

Considerable work has been done in the area of automatic translation of Chinese.[7,12] These efforts have been only partially successful. Even if a good translation device were available, however, the formidable problem of encoding Chinese characters for input would remain.

One answer to the problem would be the development of a practical Chinese character recognition machine, toward which the effort of this project is directed.* It is hoped that advances in this area would provide additional incentive for work in translation devices. On a more modest scale, a Chinese character recognition device could be used as a type of on-line dictionary to speed up the process of human translation. Even this limited application would be a welcome advance.

THE STRUCTURE OF CHINESE CHARACTERS

Chinese is a pictorial and symbolic language which differs markedly from written Western languages. The characters are of uniform dimension; they are generally square; they are not alphabetic.

The characters possess a great deal of structure and hence are well suited to the method of recognition outlined above. Many regularities of stroke configuration occur. Quite frequently, a character is simply a two-dimensional arrangement of two or more simpler characters. Nevertheless, the system is rich; strokes and collections of strokes are combined in many different ways to produce thousands of different character patterns.

Chinese characters consist of sets of connected strokes. Each stroke is roughly a vertical, horizontal, or diagonal straight line segment. Sets of connected strokes form units hereafter referred to as components. Each character consists of an arrangement of disjoint components. Figure 1 shows a character having three components.

The structure of a Chinese character may therefore be specified on two levels:

(1) A description of the internal structure of each component, and
(2) A description of the arrangement of components in two dimensions.

Components

Two questions needed to be answered when deciding how to describe the internal structure of a component:

(1) What class of objects should be considered as the basic picture element?
(2) What sort of structure should be used to describe the relationship between elements?

Three criteria were used in answering these questions:

(1) The structure should be easy to generate from the original pattern.
(2) It should be easy to generate a unique numeric code from the structure.
(3) The structure should represent the pattern in a natural manner.

A natural method of representing the internal structure of a component would be in terms of strokes. This indeed is the approach taken by several previous recognition schemes.[5,8] These schemes make use of on-line input, in which strokes are drawn one at a time. The difficulty with this approach for *printed* characters is that strokes do overlap and are not easily isolated. Also, the description of the relationship between strokes becomes complex.*

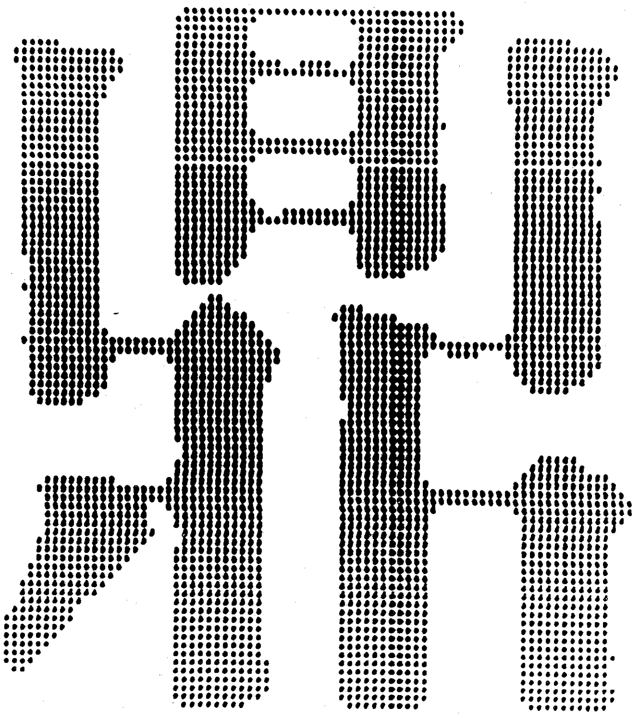


Figure 1—Character with three components

* The author is aware of one previous attempt at the recognition of printed Chinese characters.[1]

* For a discussion of a system for the description of Chinese characters in terms of strokes, see Fujimura and Kagaya.[3] The authors are primarily interested in computer generation of Chinese characters.

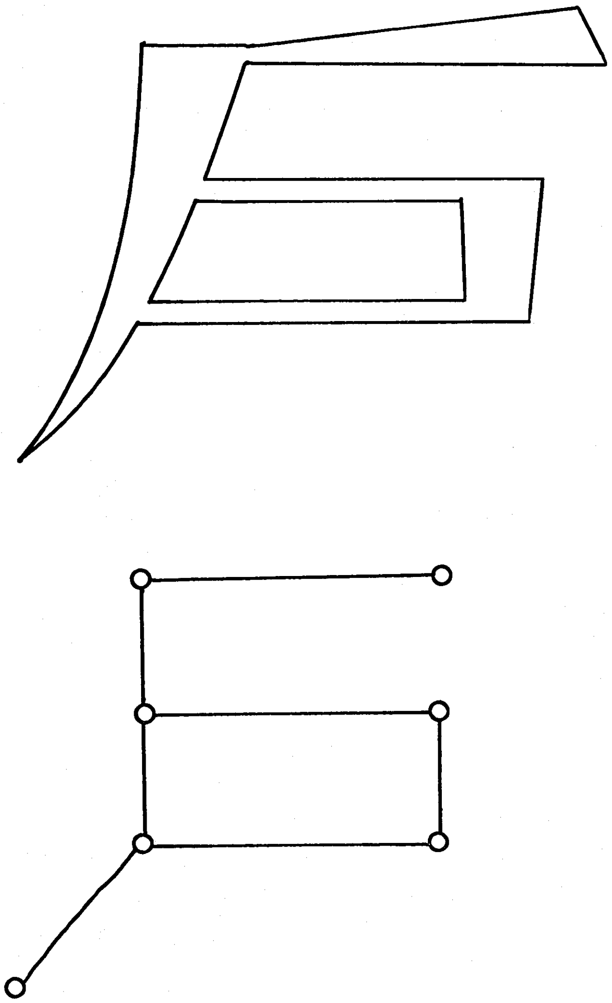


Figure 2—Component and graph

It is more promising to describe components in terms of stroke segments. This can best be understood with reference to Figure 2. As can be seen, a component can be depicted as a graph. The branches of the graph correspond to segments of strokes. These segments are bounded by stroke intersections and ends of strokes.

It will be shown later that this representation satisfies criteria (1) and (2). That it satisfies criterion (3) is fairly clear.

Characters

The arrangement of components in two dimensions to form characters can be described using the concept of frame. Each character is viewed as occupying a hypothetical square. The segmentation of a character into components segments its square accordingly. The square, or frame □, may be segmented in one of three ways: (a) East-West ◫, (b) North-South ⊟, (c) Border-Interior ▣. Each of these segmentations corresponds to a two-component character. For example, 引 would be represented by (a), which decomposes the character into 弓 and |. 台 would be represented by (b). Finally, either partial or complete enclosure, such as 句 and 困 would be represented by (c). Frames for characters composed of more than two components are obtained by embedding (a), (b), or (c) in one of the sub-frames of (a), (b), or (c). The process of embedding is recursive, in that any sub-frame of a derived frame may be used for further embedding. For example, the four-component character of Figure 3 can be described by the frame arrangement of Figure 4a. The frame description can be conveniently represented by a tree, as indicated in Figure 4b.

This description of the arrangement of components is based on the work of Rankin,[10] who introduced the concept of frame-embedding. The definition of component used here is slightly different from that of Rankin. Despite this, Rankin's claim that the three relations used in his scheme are sufficient to describe accurately virtually all characters seems to apply.

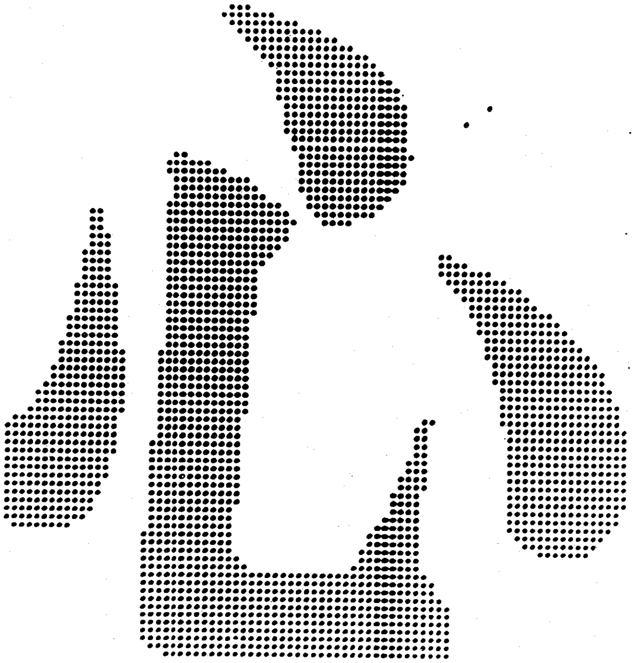
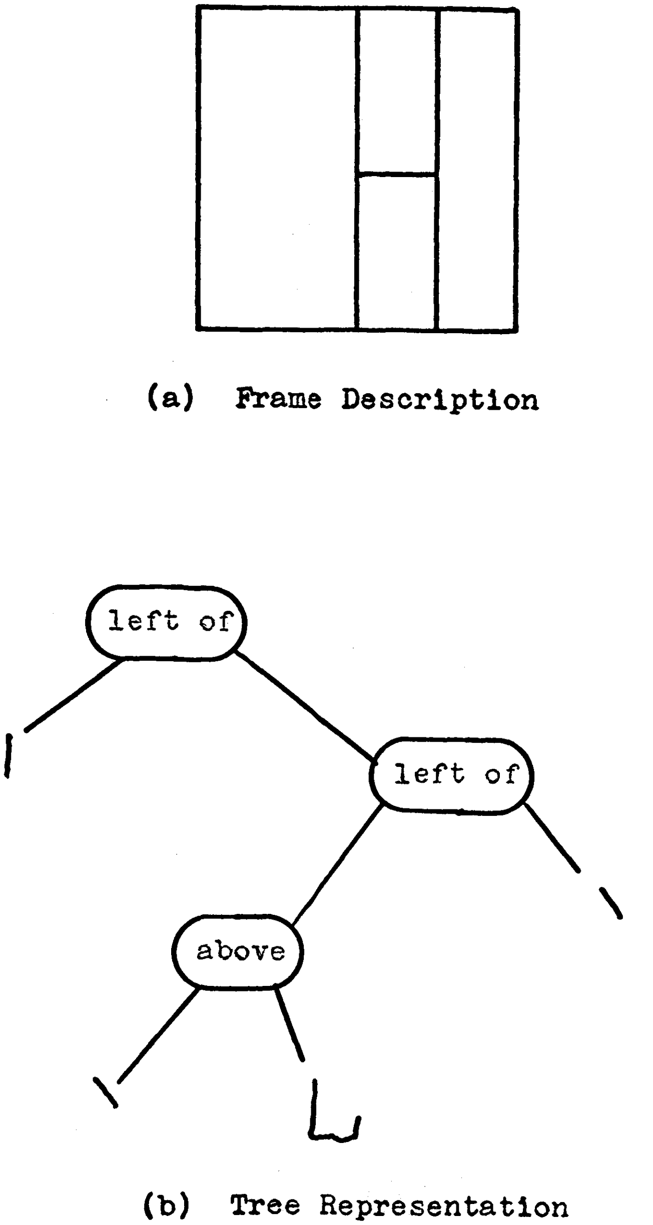


Figure 3—Chinese character



(a) Frame Description

(b) Tree Representation

Figure 4—Frame description and tree representation

INPUT

The program operates on a representation of one character at a time. The representation is in the form of a matrix whose entries have value zero or one corresponding to white or black on the original printed picture.

The device used to obtain the matrix is a flying spot scanner which measures the intensity of light reflected from an opaque sheet of paper at a number of points inside a small area or "window" on the sheet. If the intensity reflected relative to the incident intensity at a point is below a certain threshold, the value one is transmitted for that point (BLACK), otherwise a zero is transmitted (WHITE). The effect is that of placing a grid over the picture and making each square either all black or all white.

A program has been written which operates the scanner and stores the resulting matrix in core. The size of the matrix is 80×80 for a single character. However, both the window size and the sampling rate (the fineness of the grid) may be varied so that characters of varying size can be processed. Additionally, the threshold may be adjusted to achieve the best quality.

Certain functions of the program depend on the fact that there are no gaps or holes in any of the strokes. This is not always the case due to the quality of the printed input. Accordingly, a smoothing operation is performed to fill in the gaps. The resulting matrix is used as the data base for the program.

The digitized form of a character can be displayed on a CRT. Figures 1, 3, 5 and 6 are photographs of such displays.

ANALYSIS OF COMPONENTS

A program has been written to perform the analysis of components. For a given component, the output of the program is a connected graph in which branches correspond to stroke segments and nodes correspond to the endpoints of stroke segments.

The program is quite complex, and will be described in detail in a future paper. A brief outline of the procedure is given here.

The procedure begins by finding an arbitrary stroke segment of a component. This is done by scanning along various rows and columns of the matrix until a stroke is encountered (See Figure 5). This initial stroke segment is the first branch of the graph. Next, the two endpoints of this initial segment are found. This is done by crawling along the stroke in both directions. Crawling is accomplished by following along the boundary points of the stroke using a standard contour-tracing algorithm. The crawling halts either at an intersection of strokes, characterized by encountering a large black area, or at the end of a stroke. Both of these two conditions for halting correspond to a node of the graph being encountered.

Thus, one initial branch and its two nodes are found. The procedure continues in the same manner. For each of the two nodes, all segments leading from it are investigated (by crawling), finding more nodes. Segments leading from each new node are similarly investigated until the entire component has been covered. During the execution of this recursive procedure, the graph of the component is developed as each new node is encountered.

ANALYSIS OF CHARACTERS

The algorithm for analyzing the pictorial structure of a character is in two parts:

1. A collection of connected graphs is produced, one for each component.
2. The relationship among components is determined.

Finding all components

The program described in the previous section must be applied to all the components of a character. Some method must be used for finding each of the components and keeping track of those which have been analyzed.

To do this, the following procedure is employed. As a component is being analyzed, its outline is drawn on a separate pattern. That is, the contour points of a component are filled in on a new pattern as they are encountered during analysis. The auxiliary pattern contains, at any time, the outline of all the components of a character which have been processed.

After a component has been processed, a search is made for a stroke of a new component. If a stroke is found, its boundary points are checked against the auxiliary pattern to determine whether or not this stroke belongs to an already-analyzed component. In this way, new components may be found and analyzed. The procedure halts when no new strokes can be found. The result is to produce a collection of connected graphs.

Figure 6 shows the result of applying the algorithm to the character of Figure 3.

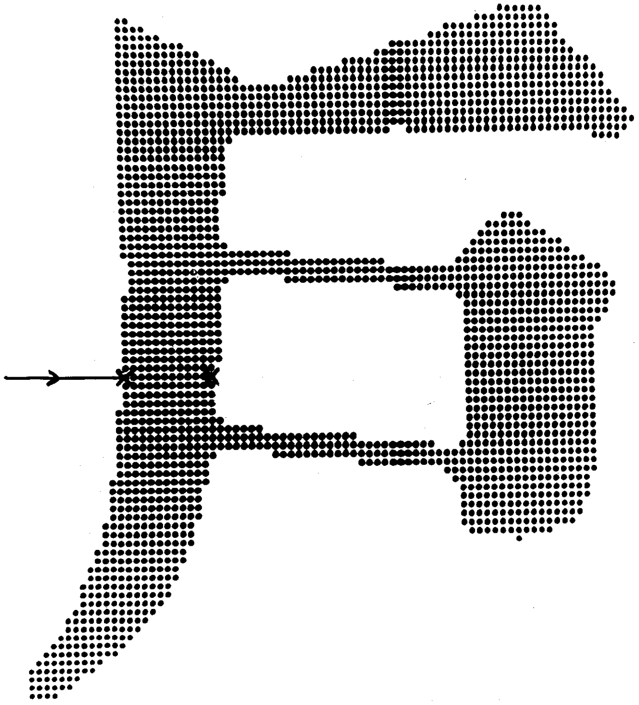


Figure 5—Finding a stroke

Constructing the frame

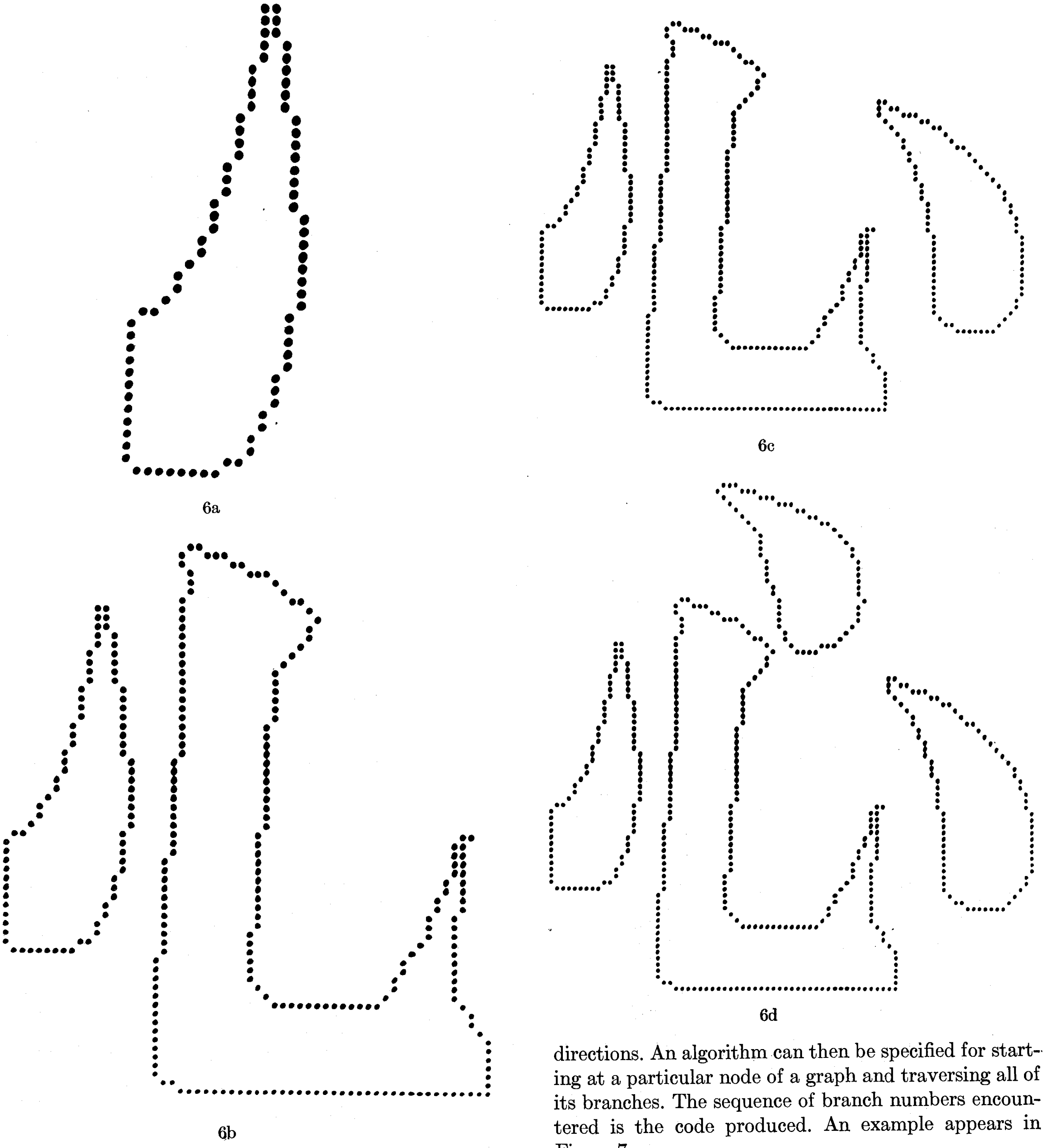
Representation of the frame description of a character is done conveniently by means of a tree. The root node of the tree has as its value one of the three relations indicating how the overall frame is broken into two sub-frames. The two sons represent the structure of the two sub-frames. Terminal elements correspond to components (see Figure 4b).

The method of obtaining such a tree will be briefly described. First, each component in the character is inscribed in a rectangle. This is easy to do since the coordinates of each node are known. The relationship between all possible pairs of components is determined by determining the relationship between their rectangles. The one of the three permitted relationships (East-West, North-South, Border-Interior) which most nearly approximates the true relationship is chosen. Then it is determined if one of the components has the same relation to all other components. This will usually be the case. If so, that component becomes one son of the root node of the tree; the value of the node is the appropriate relation; the other son is a tree representation developed for the remaining components. This sub-tree is determined in the same way.

If no single component is found, a more complicated procedure is used to determine if any two components have the same relation to all others, and so on.

ENCODING OF COMPONENTS

For recognition purposes, a procedure has been developed for generating a numeric code for each character. The first step in this procedure is the generation of a code for each component in a character.



6a

6b

6c

6d

Figure 6—Outline of a character

The code for a component is generated from its graph. To this end, the branches of a graph are labeled at each end. The label on a branch at a node indicates the direction or slope of that branch quantized into eight directions. An algorithm can then be specified for starting at a particular node of a graph and traversing all of its branches. The sequence of branch numbers encountered is the code produced. An example appears in Figure 7.

The algorithm obeys the following rules:

1. Start at the node in the upper left-hand corner of the graph. Exit by the branch with the lowest-valued label. Mark the exiting branch to indicate its having been taken, and write down the branch label.

2. Upon entering a node, check to see if it is being visited for the first time. If so, mark the entering branch to indicate this.
3. Upon leaving a node, if there are available unused directions other than along the first entering branch, choose the one among these with the lowest-valued label. Leave by the first entering branch only as a last resort. Mark the exiting branch to indicate its having been taken and write down the label on the branch.

Since at each node there are just as many exiting branches as entering branches, the procedure can only halt at the starting node. At the starting node, all exiting branches have been used (otherwise the procedure could have been continued), hence all entering branches have been used since there are just as many of these. The same reasoning can be applied to the second node that is visited. The first entering branch is from the starting node and this branch has been covered both ways. But this branch would only have been used for exit from the second node if all other exits had been exhausted. Therefore all branches at the second node have been covered both ways. In this manner, we find that the branches of all nodes visited have been traversed both ways. Since the graph is connected, this means that the whole graph has been covered.

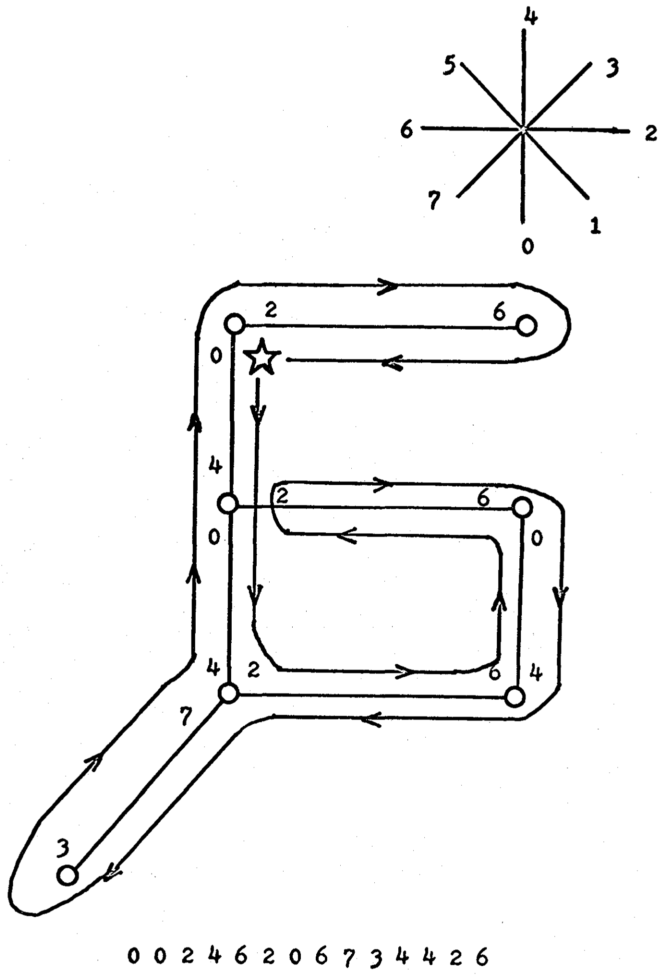


Figure 7—Encoding a graph

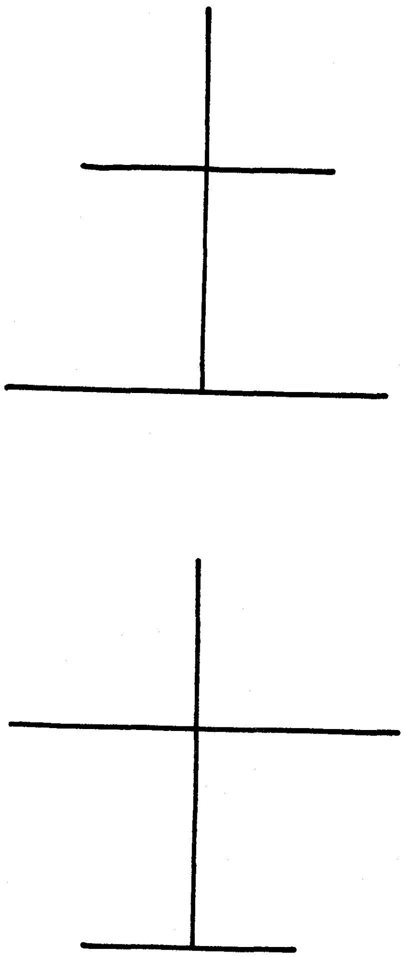


Figure 8—Two characters with same graph

All branches are traversed exactly once in each direction by this procedure, so all labels are picked up. The code consists of the branch labels in the graph written down in the order in which they are encountered.

This algorithm is based on a procedure for traversing graphs described in Ore.[9]

While it is true that this scheme will always generate identical codes for identical (isomorphic with the same labels) graphs, the goal of generating a unique code for each character is not achieved. Two types of difficulties are encountered.

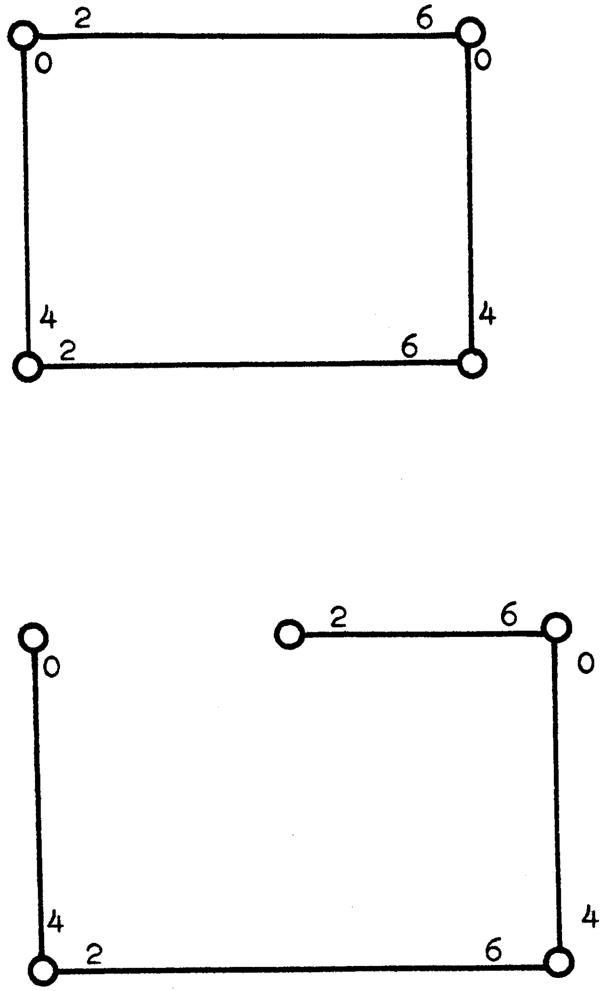


Figure 9—Two graphs with same code

The first is that two characters might generate the same graph, hence the same code. Figure 8 gives an example of such a situation. This situation appears to be very rare, however, and it seems to be no great hindrance to make special cases of these characters.

The second difficulty is that two different graphs might generate the same code. Figure 9 illustrates this situation. The author has been unable to find any pair of characters whose graphs exhibit such a property. Again, this appears to present no great problem.

The method, then, will generate different codes for different characters with only a few exceptions. The algorithm is simple and its execution is quite rapid.

ENCODING OF CHARACTERS

The representation of a character is in the form of a tree. The nodes of the tree are binary relations; the terminal elements correspond to components. Considering the relations as binary operators, the tree can easily be flattened to prefix form. This is done by walking around the tree counterclockwise, starting from the root node, and picking up nodes and terminals the first time they are encountered. As is well-known, the string generated in such a fashion is unique; the tree can readily be reconstructed from it. To generate a *numeric* string, the following code can be used:

0↔terminals (components)
1↔left node
2↔above node
3↔surround node.

Figure 10 shows the generation of code from the tree of Figure 4.

We can consider that the code so generated defines a class of Chinese characters all of which have the same frame description. Therefore, a Chinese character may be specified by first giving its frame description code and then giving the code for each of the components

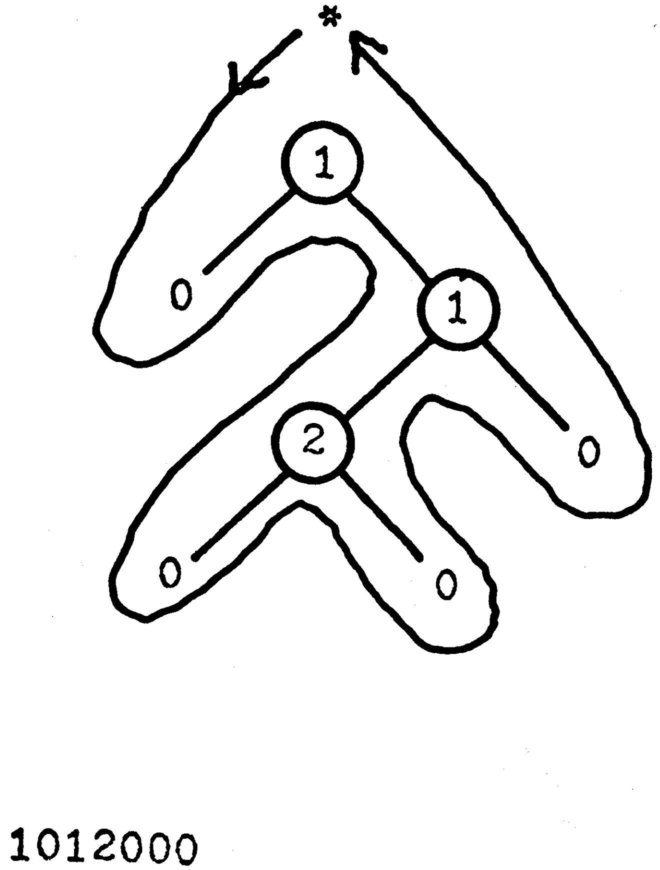


Figure 10—Flattening a tree

that fits into one of the sub-frames. A character having n components will have a code consisting of the concatenation of $n+1$ numbers:

$$N_0, N_1, \ldots, N_n$$

where N_0 is the code generated from the tree and N_1 through N_n are the codes of the components listed according to the order in which the components were encountered in the tree flattening.

RESULTS

The algorithms discussed in this paper have been implemented as a computer program which recognizes printed Chinese characters.

Most of the program is written in FORTRAN. A considerable fraction, however, is written in assembly language. The assembly language routines augment the power of FORTRAN to permit list-processing operations and also to permit FORTRAN subroutines to be called recursively. The combination, while less than ideal, provides for the creation and manipulation of complex data structures in a fairly natural manner.

The program runs on a PDP-9 computer with a core memory of 32K words of 18 bits. The computer also has a large auxiliary disk and magnetic tape storage facilities.

The program has been tested with a number of characters from several different sources. The tests were designed to consider four questions:

1. How successful is the program in analyzing the structure of Chinese characters?
2. Does the program generate consistent codes for characters of the same font? That is, will two instances of the same character from the same source yield the same code?
3. Does the program work for characters from different sources?
4. Do factors such as character size and character complexity affect program performance?

Initial results were obtained from a set of characters obtained from a Taiwan printer. A sample of this set appears in Figure 11. To start, 225 different characters were processed. This was to provide a dictionary for later tests, and to test the pattern analysis capabilities of the program.

The results show a reasonable structural representation produced for about 94 percent of the characters. The failures were all due to a particular component not being analyzed; for all characters the relationship among components was correctly determined. The problems all occurred in the part of the component analysis algorithm concerned with analyzing nodes; i.e., the part which is to find a node and locate all stroke segments leading from it. The routine would sometimes make mistakes if, for example, two nodes were very close together or one node covered a large area. The characters involved were typically quite complex.

文⋮文
斗⋮斗料斜
斤⋮斤斥斯新斷
方⋮方於施旁旅族旋旗
旡⋮既
日⋮日旦旨早旱旬昂昆昌昏明
映春昨是時晉晚晤晨普景
暑暖暗暢暴暫暮曆暨曉昭
曰⋮曰曲更書曼會替最會曹
月⋮月有朋服朗期朝望臘

Figure 11—Example of character set

From the characters that were successfully analyzed, 25 were chosen for additional testing. Four additional instances of each character from the same source were processed, for a total of 100 new characters.

All new instances of the 25 characters produced reasonable structural representations. For five of the characters, one of the new instances produced a slightly different representation, hence a different code. No character generated more than two codes. In all cases, the discrepancy was caused by the fact that two strokes which were very close in one instance touched in another instance of the same character.

Additional testing was done using two other sources. Characters from issues of a Chinese magazine were used. These were approximately half the size of the characters in the original set. Also, some computer-generated characters[6] were used. These were about double the size of the originals. Both were of about the same style. Fifty instances were taken from each source. The percentage of instances generating the same code as the corresponding character from the original set was 89 percent for the magazine source and 95 percent for the computer source. Discrepancies mostly had to do with stroke segments appearing at somewhat different angles and with strokes touching in one case but not the other.

CONCLUSIONS

Pattern description

A descriptive scheme for the structure of Chinese characters has been proposed and a program for computer analysis conforming to the scheme has been written. The description is on two levels: The internal structure of components, and the relationship among components.

The first level of description is straightforward: a connected part of a character is represented by a graph. This representation is adequate for the description of components; it is reasonable for the human percipient to think of components as graphs.

Analysis on this level works fairly well; difficulty is encountered with some complex characters. Some work has been done on modifying the described approach. The modification consists of "shrinking" a component to a skeleton and obtaining the graph from the skeleton. This procedure is sensitive to contour noise, and it seems that use of this method would result in many components generating several different graphs from different instances.

The second level of description is based on the work of Rankin. With the exception of a very few characters whose components do not fit neatly into the frame description, it is an effective means of describing the structure of Chinese characters in terms of components. The analysis program for this level has been successful for all characters tested.

Pattern recognition

It would be overly optimistic to claim that the results of this thesis prove the feasibility of a hardware Chinese character recognition device. The development of such a device would be an impressive accomplishment. Nevertheless, the author feels that this thesis points the way to such a device by providing a good method for encoding Chinese characters.

There are two separate, but related, problems associated with Chinese character recognition. The first is that complex characters may fail to be encoded, or may generate several codes. The second is that there would need to be a quite large dictionary of characters to handle most material. A college graduate in China is supposed to know about 5000 characters, which gives some indication of what would be needed. The problem of handling a large dictionary arises. This is especially true if many of the characters require more than one code.*

It is likely that a recognition device would be used to process material from mainland China. If this is true, several developments make things look brighter. They result from the desire of the Communist Government to simplify the language.[2] The first is that the Government has recommended the general use of only 2000 characters. A list of these characters has been published. Publishers, being Government-controlled, are under instructions to stay within the total of 2000 as far as possible. Secondly, the Government has simplified a large number of characters. In 1956, a list of 515 simplified characters to be used in lieu of the original complex forms in all publications was released. The average number of strokes per character for the 515 was reduced from 16 to 8. Overall, since 1952 the average number of strokes per character has been reduced from around 13 to around 10 for the 6000 most frequently used characters. The continuing Communist policy of language simplification contributes to the author's opinion that a Chinese character recognition device is a realistic objective.

ACKNOWLEDGMENTS

The author would like to thank Professor Francis Lee of M.I.T., whose guidance and advice were invaluable to this project. The author is also grateful to Professor Thomas Huang of M.I.T. and Professor Herbert Teager of Boston University for their help. This paper is based on a thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical Engineering at the Massachusetts Institute of Technology on August 15, 1971.

* One way to reduce the number of characters requiring more than one code would be to use a stylized font especially designed for use with a character recognition device.

REFERENCES

1 CASEY NAGY
Recognition of printed Chinese characters
IEEE Transactions on Electronic Computers
February 1966 pp 91–101

2 Y CHU
A comparative study of language reforms in China and Japan
Skidmore College Faculty Research Lecture
Skidmore College Bulletin December 1969

3 FUJIMURA KAGAYA
Structural patterns of Chinese characters
Research Institute of Logopedics and Phoniatrics
U of Tokyo Annual Bulletin #3 April 1968–July 1969
pp 131–148

4 U GRENANDER
A unified approach to pattern analysis
Advances in Computers Volume 10 Academic Press 1970
pp 175–216

5 GRONER HEAFNER ROBINSON
On-line computer classification of handprinted Chinese characters as a translation aid
IEEE Transactions on Electronic Computers
December 1966 pp 856–860

6 A V HERSHEY
Calligraphy for computers
U S Naval Weapons Laboratory 1 August 1967
AD 622 398

7 KING CHANG
Machine translation of Chinese
Scientific American June 1963 pp 124–135

8 J H LIU
Real time Chinese handwriting recognition machine
EE Thesis M. I. T. September 1966

9 O ORE
Theory of graphs
American Mathematical Society 1962

10 RANKIN TAN
Component combination and frame-embedding in Chinese character grammers
NBS Technical Note 492
February 1970

11 K M SAYRE
Recognition: A study in the philosophy of artificial intelligence
U of Notre Dame Press 1965

12 L YUTANG
Final report on Chinese-English machine translation research
IBM RADC-TDR-63-303 10 June 1963

Computer diagnosis of radiographic images*

by S. J. DWYER, III, C. A. HARLOW, D. A. AUSHERMAN and G. S. LODWICK

University of Missouri-Columbia
Columbia, Missouri

INTRODUCTION

The potential of optical scanning equipment and digital computers for assisting or replacing human judgment in medical diagnosis has been recognized by investigators for some time.[1] A number of efforts have been made, with varying degrees of success, in developing automatic techniques for recognizing and classifying blood cells,[2] chromosome analysis and karyotyping,[3] identifying leucocytes,[4] and processing scintigram images[5] obtained in nuclear medicine. These image analysis techniques are now being extended to the clinical specialty of diagnostic radiology where there is an urgent need to provide assistance in handling the several million radiographs read by radiologists each year. The need for computer-aided diagnosis in radiology is becoming increasingly urgent because of the expanding population and the continuing demand for improved quality of medical care. The use of computers in radiology can free the diagnostic radiologist from routine tasks while providing more accurate measurements that lead to consistent and reliable diagnoses.

The most common radiological examination is the conventional chest film. The resultant film is a complex image consisting of a two-dimensional projection of a three-dimensional object, whose structure reflects the absorption of X-rays as recorded on film. Such films commonly offer photographic resolution of 40 line pairs per millimeter. Chest films are commonly called for in routine medical examinations and are used by physicians to determine the health status of the patient. In the case of coal workers' pneumoconiosis ("black lung disease"), chest films are even employed as the basis for determining federal compensation to the individual workers. The chest film examination is employed by the radiologist to ascertain diseases such as lung disease, rheumatic heart disease, congenital and acquired heart disease, skeletal anomalies, and radiation-induced diseases. In addition to the conventional plain films of the chest, other radiant images are employed in radiographic diagnostic techniques, such as a cardiac series, fluoroscopy, image amplification, video tape, and cine radiography. This paper presents the results of an automatic visual system for the diagnosis of heart disease from the conventional radiographic technique employed to form the plain film of the chest.

The basic technique involves the conversion of the chest film to a two-dimensional array of numbers, each number in the array representing the transmittance of light through the film at a particular point on the film. Analysis is then carried out by pattern recognition methods on the digitized image as implemented by the digital computer. Four steps comprise the automatic visual system for diagnosis of heart disease from the chest film.

The first step is the digization of the X-ray image. A specialized camera scans the film which is illuminated from the back, producing an output voltage whose average value is proportional to the amount of light transmitted through a particular point on the X-ray. Each such point is selected by the computer directly controlling the scanner camera. The optical image is thus converted point by point into a two-dimensional array of numbers, each of which represents the "grayness" of the image—that is, the gradation of light and dark on the film—at a particular point.

The second step in the process is that of preprocessing. These techniques are designed to enhance the extraction of selected features and to eliminate irrelevant data in an image. A number of these techniques have proven useful. Gray level histograms are employed to determine if a distribution transformation should be employed. Spatial digital filtering methods utilizing the fast Fourier transform or recursive partial dif-

* This work was partially supported by USPHS Grant GM 17729.

ference equations may be useful. Contrast enhancement is often employed, especially if the digital processed images are to be displayed. Finally, image subtraction is often used to remove irrelevant image information.

The third step is that of feature extraction. The actual selection of an image feature set is a significant problem in itself for which no general theoretical solution exists. One reason for this difficulty is that a set of features required for classification—normal or abnormal samples for a particular disease—is relative not only to the class of images but also to the diagnostic problem under consideration. Also, features needed for one diagnostic problem are useless for another problem. The most difficult portion of an automated image analysis system is that of feature extraction. Techniques which are useful in the feature extraction portion of the process are directional signatures, contour tracing, Fourier transform frequency signatures, template matching, region enumeration techniques, and linguistic methods.

The fourth, and final, step in the process is that of automatic classification. Successful use of pattern recognition techniques requires that all four steps in the process be carefully designed, depending upon the specific problem to be automated. In the case of chest films, classification schemes that have proved useful consist of discrimination functions, both for the normal-abnormal two class problem and for the differential diagnosis—the further division of the abnormal class into subclasses of diseases.

DIGITIZATION OF THE CHEST X-RAY FILMS

Chest films are scanned with a computer-controlled specialized camera that produces an output voltage whose average value is proportional to the amount of light transmitted through a particular point on the X-ray.

The system as depicted in Figure 1 performs three categories of operations: (1) the conversion of images in the form of film transparencies into digital arrays of numbers; (2) the storing of these arrays in an orderly fashion to facilitate retrieval for use by researchers; and (3) the redisplay of these and other picture arrays either to produce a photographic copy or for direct viewing. The conversion of film to numbers is accomplished either by a flying spot scanner (FSS) or an image dissector scanner (IDS) which utilize an image dissector camera in the digitization. Both of these devices operate under the control of a digital computer which also handles the flow of data from scanner to magnetic tape unit.

The digital images are recorded on 800 BPI magnetic tape during the scanning process, becoming permanent entries in the image library. The library consists of digital images stored on IBM-compatible magnetic tape in a special library format that facilitates locating and using every entry in the library. An identification record containing scanning parameters and content information is written at the beginning of each image on a library tape. A library index in punched card form contains identical information and is updated as new images are recorded.

The third function which the system performs is the reconstruction of real images from digital form. Three separate devices accomplish this. The first device is a Hewlett Packard 1300A X-Y display scope with a x-axis input. Second is a Dicomed/30 Image Display, manufactured by Dicomed Corporation of Minneapolis. The third device is a digital image display utilizing a high speed disc memory which is filled at slow data rates, and then dumped at high data rates as video information to a high resolution, high tonal television monitor display. Another distinguishing feature of this device is its interactive capabilities. Through the use of a joystick the operator may extract certain features of the displayed image and enter these into the computer.

Central to the operation of both scanning and display devices, the SEL 840A digital computer acts as controller and information channel for the system. The SEL 840A is a small scale computer with a memory cycle time of 1.75 microseconds and 16K of word memory. A minicomputer could also perform the necessary tasks. An IBM compatible tape drive is necessary for library creation.

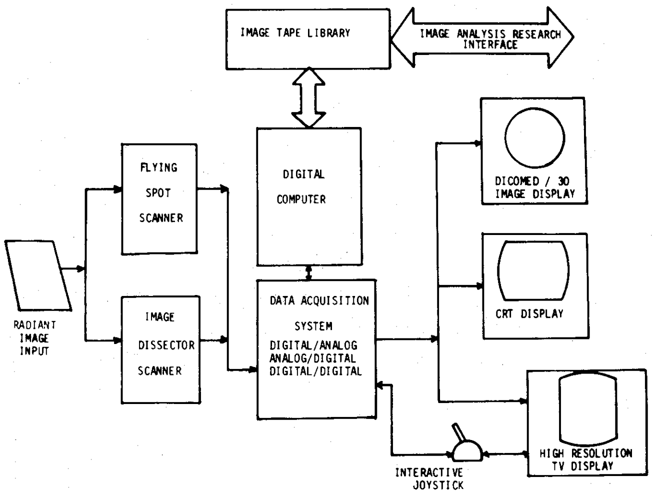


Figure 1—Digital image scanning, storage and display system

Digitization devices

Two digitization devices have been employed in the digitization of chest films—the flying spot scanner (FSS) and the image dissector scanner (IDS).

Flying spot scanner

The flying spot scanner has seen wide use as an image digitizer. Serving as two devices in one, the FSS can also reconstruct hard copy photographic images from digital form. For this reason the FSS is sometimes chosen as both the digitizing and reconstruction instrument in many image analysis applications.

The basic components of an FSS system are shown in Figure 2. When the transmittance at a pair of particular coordinates on the film is to be measured, a dot is illuminated on the CRT at a corresponding location. The position of the dot is controlled by X and Y deflection voltages which are usually generated by digital-to-analog converters. These converters are an integral part of the SEL Computer Data Acquisition System which operates as a peripheral device to the SEL 840A. An image of the computer-positioned dot of light is focused upon the film by the lens. The amount of light passing through the film is measured by a photomultiplier tube positioned behind the film. This signal contains some error due to the fact that the brightness of the dot is not entirely uniform over the surface of the CRT. To account for this error the beam is interrupted by a beam splitter and the intensity of the dot is measured by a second photomultiplier. Both signals from the photomultiplier tubes contain a good deal of electron shot noise. This must be removed by extensive filtering prior to analog-to-digital conversion of the signals. The ration of transmitted signal to reference signal, calculated either digitally or by analog techniques, is the the quantity of interest. This measurement is repeated rapidly for all of the locations in the picture raster.

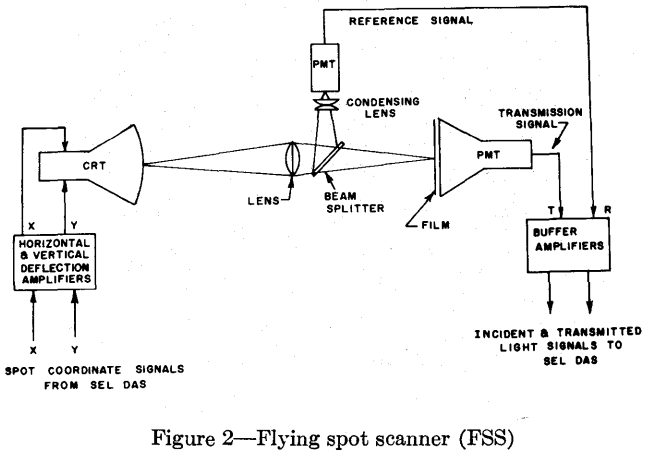

REFERENCE SIGNAL
PMT
CONDENSING LENS
TRANSMISSION SIGNAL
CRT
PMT
LENS
BEAM SPLITTER
FILM
T
R
X
Y
HORIZONTAL & VERTICAL DEFLECTION AMPLIFIERS
BUFFER AMPLIFIERS
X
Y
SPOT COORDINATE SIGNALS FROM SEL DAS
INCIDENT & TRANSMITTED LIGHT SIGNALS TO SEL DAS

Figure 2—Flying spot scanner (FSS)

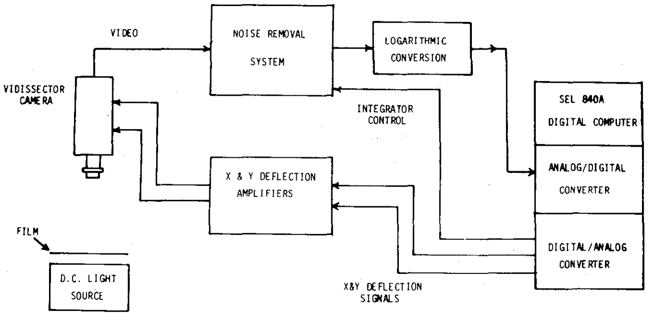


Figure 3—Image dissector scanner system (IDS)

To reconstruct photographs from digital images unexposed film is placed in the FSS and the dot intensity and position are controlled so as to expose the film. The reference photomultiplier measures the amount of exposure that the film is receiving.

Factors affecting the resolution or ability to delineate fine detail of a flying-spot scanner system include CRT spot size, lens resolution, film image contrast and sharpness, and phototube signal-to-noise ratio (SNR). The resolution obtained in an FSS system seems to be proportional to the cost spent in constructing the device. An inexpensive, low resolution FSS has been implemented and will scan 3″ × 3″ film size at a limiting resolution of 256 samples across the film. This has proven quite sufficient for low-resolution image analysis applications. Very high resolution has been attained elsewhere but at a tremendous increase in cost. A disadvantage of the FSS system is that it is difficult to scan large format films such as full size medical radiographs.

Image dissector scanner

The image dissector scanner is the second digitization device utilized. The distinguishing component of the IDS system as depicted in Figure 3 is the vidissector camera, manufactured by International Telephone and Telegraph Industrial Laboratories. The camera employs an image dissector tube to sample the input image.

The image dissector has been described as a photomultiplier with a small electronically movable photocathode area, thus acting as an all-electronic low-inertia microphotometer. The image to be scanned is focused by the camera lens onto a 1.75 inch circular photocathode. Electrons are emitted from the back of the photocathode, forming an electronic image with current density modulated according to the image input. The electron image, focused by magnetic deflection, falls upon an aperture plane, at the other end of the drift tube. The aperture "samples" the image by allowing only a small, well defined area of the electron image to pass through. The sampled photoelectrons are then multiplied by an electron multiplier by a factor of approximately 5 × 10^5. The entire electron image is deflected allowing the aperture to sample different points in the picture.

The input image is a chest radiograph measuring 13½ by 13½ inches and illuminated from behind by a DC powered, high intensity light source. The camera deflection signals are generated by the SEL computer and converted to analog in the SEL data acquisition system. Deflection buffer amplifiers are used to adjust the amplitude and offset of the deflection signals. Eleven bit DAC's are used to allow for scanning rasters of 2048 × 2048 points, although this actually exceeds the resolution of the camera.

The noise current present in the video output of an image dissector camera system can be attributed to random fluctuations of photo-electrons entering the aperture, modified by the statistical noise properties of the electron multiplication process. It is necessary to remove this noise prior to analog-to-digital conversion. This is accomplished by integrating the video signal for a suitable length of time. The resulting signal-to-noise ratio becomes a function of tube construction, input image brightness, and integration time. The integrate time can be selected, but 800 microseconds is a common time. This relatively long integrate time gives a S/N ratio of over 40 DB for a relatively bright input image. This has been verified by digital frequency analysis of the video signal.

The resolving capabilities of an image dissector are determined by the shape and size of the dissecting aperture, usually a circular aperture of diameter 1 mil. This configuration gives a resolution modulation amplitude of 39 percent at 1000/TV lines per inch on the photocathode. The usable diameter of the photocathode is 1.4 inches, given a resolution of 1400 TV lines across the image diagonal.

Logarithmic conversion via a logarithmic video amplifier allows recording of the optical density of a film, rather than transmittance. This is an optional feature, allowing better gray shade condition in the darker areas of a film. An 11 bit analog-to-digital converter is used for the video signal, a more than sufficient number for gray-shade rendition. However, this allows less critical adjustment of the video signal amplitude to insure sufficient quantization levels.

A definite advantage of the IDS system over the FSS system is its ability to scan large format films, such as 14 inch by 14 inch radiographs. With proper optics the IDS can scan small format also. A disadvantage of the IDS system is the inherent non-uniformities in the illuminating light source. Cathode non-uniformity can be typically ± 15 percent over the usable surface, while light source non-uniformity depends upon construction of the source. In the FSS non-uniform CRT response could be compensated for in real time by the reference photomultiplier. For the IDS the correction had to be made by scanning a blank film and then subtracting or dividing each subsequent scan, point by point, by the reference image array. Subtraction was used for density scans, while division must be used for transmittance scans. Figure 4 shows the IDS.

Display devices

Two digital display devices are utilized to examine the processed image—a dark trace storage tube display and a disc memory-TV display.

DICOMED 30 image display

A display device used for examining the processed chest X-ray images is one that is designed and manufactured specifically for displaying digital images (Figure 5).

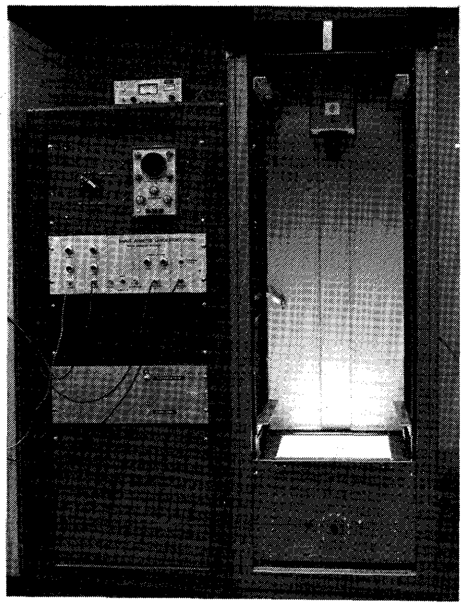


Figure 4—The image dissector scanner system

The DICOMED/30 Image Display is a product of the DICOMED Corporation of Minneapolis, Minnesota. The device interface with the computer is purely digital and is operated like a computer peripheral. The display utilizes a unique type of CRT called a dark trace storage tube which uses a scotophor instead of a phosphor to produce the image. The scotophor is deposited upon the 8 inch viewing surface of the tube. When struck by the electron beam it becomes more opaque and remains so until it is erased by thermal neutralization. This is in contrast with a phosphor CRT which glows only momentarily when it is struck by an electron beam. This almost infinite persistence characteristic of the dark trace storage tube makes it ideal for converting digital images into a usable form. Under program control the DICOMED display "paints" an entire image, one raster point at a time, upon the CRT viewing surface. When illuminated from behind by a viewing light, the CRT face portrays the reconstructed image which remains in full view without need for refreshing by the computer. Thus, with this device, the user can either view the reconstructed image directly or he may desire to make a photograph of it.

The DICOMED display has a set 1024 by 1024 square raster superimposed on the 8 inch circular viewing surface. With digital pictorial data that consists of fewer raster points only a portion of the viewing also need be utilized or, if one wishes, the smaller rasters can be interpolated at display time up to the set 1024 point raster of the display.

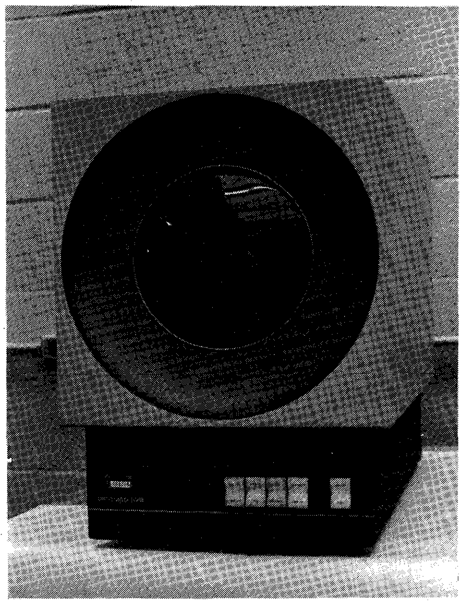


Figure 5—The DICOMED model 30 digital display

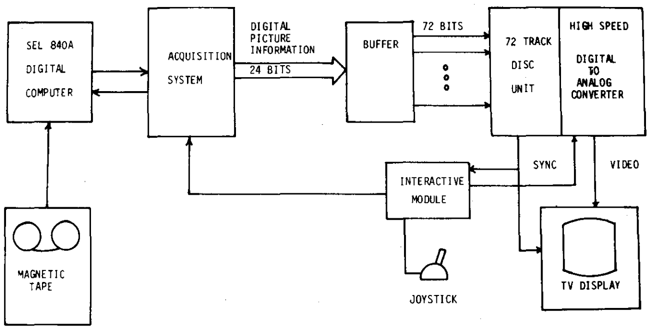


Figure 6—High speed disc image display system

The display and its associated control program can reconstruct an image from magnetic tape in approximately 100 seconds, the time being nearly the same regardless of raster size. The image that is produced has high resolution and can contain up to 64 distinct shades of gray. The contrast ratio of the display is a maximum of 3 to 1 but this can be improved upon when producing "hard copy" by using high contrast film and high contrast paper to print the resulting pictures on. The erasure of images is automatic and takes about 15 seconds to complete one erasure. This is accomplished by heating the scotophor until the scotophor returns to its neutral state. The storage property of the DICOMED Image Display makes it ideal for converting digitally coded pictorial data into a useful form.

*Disc memory-TV display**

Data transfer rates of conventional magnetic tape drives and digital I/O channels are much too slow for flicker-free CRT or TV display of high resolution digital pictures. In the high speed disc-display system, digital picture information is fed to a disc memory at these slow rates, then dumped repeatedly to a television monitor at rates high enough to produce a flicker-free image. The system operates as shown in Figure 6. The image data is read from magnetic tape at interface rates, then dumped via the SEL DAS digital I/O channel into a 72 × 16 bit buffer array. When full, the array is dumped to the disc. The disc is a Data Disc which has parallel read/write capabilities on 72 tracks of 100,000 bits per track. The disc will hold up to three 525 line pictures or one 945 line picture at a time. Once the disc has been filled with the desired picture information, the computer is free to perform other tasks, as the display then acts as a stand-alone facility. The image information is read in parallel by 72 heads, converted to analog form by high speed

* Development supported in part by NSF Grant 20401.

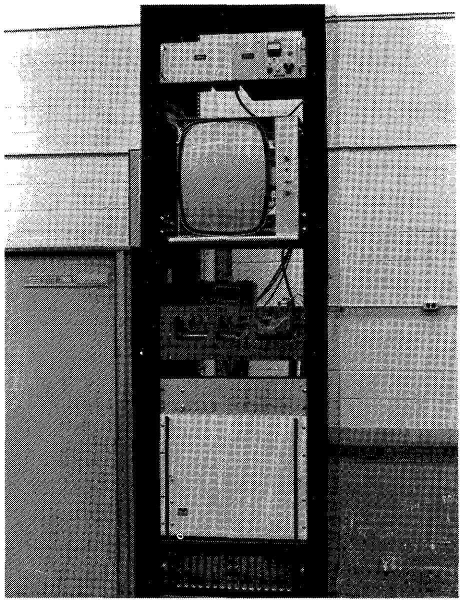


Figure 7—Disc memory-TV display

DAC's, and displayed as an image on the the television monitor at 30 frames per second with interlaced alternate frames, producing a flicker-free image. A control switch determines which of the three 525 line images on the disc is to be displayed.

The television monitor itself has a resolution of 1800 TV lines per picture width, adequate for many applications. The monitor also has a contrast ration of 10 to 1, a substantial improvement over the DICOMED display. Also, picture fill time for a 525 line image is less than 20 seconds compared to 100 seconds for the DICOMED/30 display.

One of the most distinguishing features of this image display system is its interactive capabilities. Although interactive displays are quite common in graphics applications, an interactive high tonal, high resolution image display is a rarity.

This interactive mode is accomplished by using an interactive module which employs a joystick to position a dot of light on the television monitor. The joystick generates X and Y analog voltages which are digitized and compared with the X and Y coordinates being displayed on the monitor. When a match occurs, the image information at the location is ignored and a white dot displayed instead. The X and Y coordinates of the dot may be displayed on a digital readout panel or they may be sent upon command into SEL memory via the digital channel on the data acquisition system. In this manner objects in the image field may be outlined and that outline recorded in computer memory or disc. Another use would be to take measurements of objects in the image. Figure 7 shows the display.

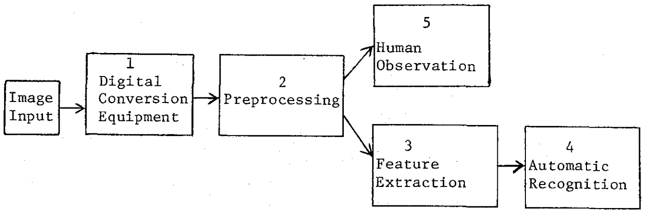
COMPUTER ANALYSIS

Once the image has been converted into an array whose values represent the gray shades in the image, the data are processed on a digital computer for the purpose of identifying and extracting the relevant features of the image. We presently use an IBM 360/50 computer in this processing. It is helpful to consider the structure given in Figure 8 while discussing an image processing system. Stage 1 in this figure has been discussed in the previous section.

In the preprocessing stage, stage 2, one can usually devise methods, such as filtering routines and other preprocessing routines, that are adequate, but not optimal, for the later stages of processing. In some of our studies on chest X-rays we have found that we need no preprocessing. In some of the studies in nuclear medicine a simple averaging technique suffices, which reduces the data to a resolution of 32 × 32 approximating the resolutions of the input camera. Some of the more useful preprocessing techniques will now be discussed.

Distribution linearization

Because image digitization requires that each gray level value be quantized over a finite range (e.g., 6 bits/



1. Digital conversion equipment - The camera and associated analogue to digital conversion equipment required to convert the image into an n x n array of points whose entries give the brightness value of the image.

2. Preprocessing - Programs used to suppress the noise put into the image data by the digital conversion equipment and also programs used to emphasize certain properties in the image that are more important than others for the later stages of analysis.

3. Feature Extraction - The extraction from the image data of information to be used in the automatic recognition of objects in the image. The data emitted from stage 3 of processing should be greatly reduced from that of stage 2 of the processing.

4. Automatic Recognition - The automatic recognition of the important objects in the image by the computer.

5. Human Observation - A human interpretation of an improved image.

Figure 8—Image processing system

picture element), a histogram of the gray level distribution values may be computed.

The gray level histogram (first order probability density function) of most images exhibits a brightness peak which is heavily biased to the dark side of the histogram.[6,7] The digital representation of the radiograph has 256 × 256 brightness values over a 33 × 33 centimeter face, with a sampling frequency in either dimension of .78 lines/mm. The effect of having a large percentage of the 65,536 picture elements (pixels) concentrated into such a narrow portion of the histogram is an image with little visible detail. Many contrast ratios that define edges are simply not displayable. One technique frequently used to correct this is the application of logarithmic conversion of the brightness pixels. This yields pixels which are proportional to image film density rather than brightness.[8] The log operation has the effect of expanding the gray range of the lower brightness pixels while compressing that of the higher brightness pixels. However, because of the shape of the brightness histogram, there is much more contrast expansion than compression.

Because a more rectangular histogram is advantageous, a position invariant, nonlinear histogram equalization technique, similar to the distribution transform in statistics[9], is useful. Since the distribution transformation is only true for continuous variables, some care must be taken to obtain the desired result for the discrete approximation.

Linear filtering

Several techniques for spatial and frequency domain design of spatial digital filters will now be considered.

Digital smoothing is designed to remove noise from images so that later processing will be made easier. Smoothing is usually done by either spatial or frequency domain operations.

One method is to average over a neighborhood in the spatial domain. The basic idea is simply to assign a neighborhood to a point and then change the intensity value of the point by performing an average of all its neighbors.[10] In addition, gradient functions may be set up in various directions in order to test whether or not the averaging process should be performed. An attempt is made to prevent blurring edges or points where significant pictorial information occurs. Noise removal depends intimately upon the nature of the pictures which are being processed. That is, what appears to be noise may be a significant feature in a particular picture.

These averaging methods can also be performed in the frequency domain. One takes the input picture in digitized form and performs the discrete Fourier transform or equivalent methods. The smoothing operations that have been discussed can be performed by removing the high frequencies in the frequency domain. This operation is called low pass filtering. Instead of merely deleting the high frequencies of a picture in order to remove noise, often a selected band of frequencies will be deleted while enough of the high frequencies are left to keep the edges clear.

When an image is low pass filtered, the image contrast is generally unaffected. However, edge detail is effectively removed.[11] Other types of filters can be developed based on a low pass prototype.

Enhancement techniques have an obvious potential for assisting roentgenographic diagnosis. The ability to sharpen detail, for example, and to employ nonisotropic filters to make a hairline fracture more evident, could result in more correct diagnoses, especially when combined with high resolution display equipment for human viewing.

The effect of high pass filtering on an image is to remove the contrast information of the image while outlining edges. The edge outlining effect can be seen most obviously at sharp, high contrast ratio edges. A major application of high pass filters is in the visualization of small, low contrast features superimposed onto uniform backgrounds.[12,13]

A filter which partially suppresses the lower frequency components while enhancing those higher is the high emphasis filter. It should be noted that both the high emphasis and high pass filters create negative brightness pixels. These usually appear as dark bands surrounding the sharper, high contrast ratio edges in an image.

Feature extraction

Feature extraction consists of the extraction of significant features from a background of irrelevant detail. Methods for the enhancement of selected features and elimination of irrelevant detail have just been discussed. In this section, several techniques for the extraction of significant features from an image function will be described.

The selection of an image feature set is a significant problem in itself, and no general theoretical solution exists. One reason for this difficulty is that the set of features required for classification of normal or abnormal samples for a particular disease is relative not only to the class of images but also to the diagnostic problem under consideration. Also, information which is pertinent to one diagnostic problem may be irrelevant to the solution of another problem.

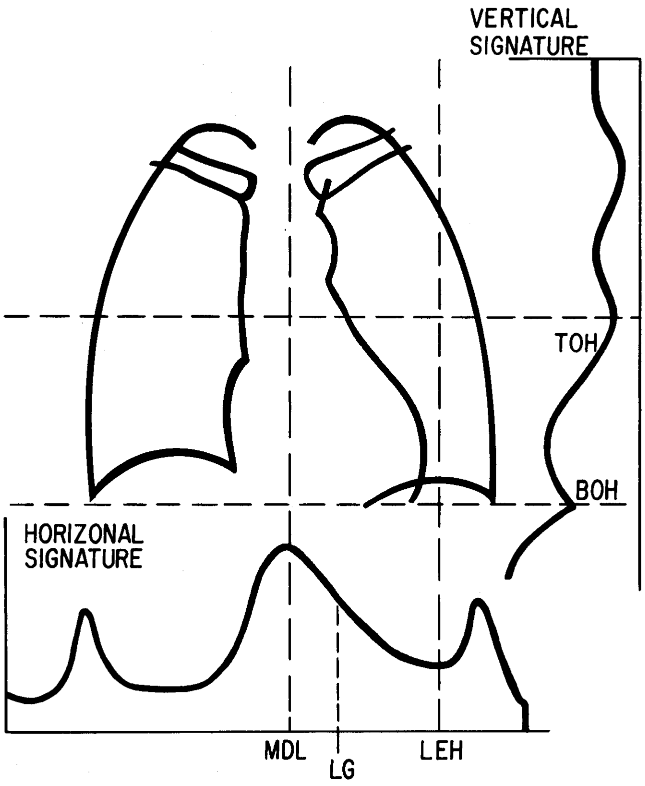


Figure 9—Directional signatures for chest radiograph

Fourier transform frequency signatures

The utility of spatial frequency power-spectrum sampling for automatically classifying patterns in images was demonstrated by Lendaris and Stanley,[14] with particular applications to detecting targets in aerial photographs. The purpose of this section is to consider the applications to biomedical images.

The usefulness of frequency sampling is based on the fact that certain features of an image function may be more distinguishable in the frequency domain than in the spatial domain. Also, a large data reduction may be affected and the features still distinguished.

The frequency signature consists of a set of n samples, where n is the number of sampling areas for a given sampling geometry. Several sampling geometries may be used. An annular ring sampling geometry is suited to the detection of circular objects. A wedge shaped geometry can be used to detect periodic line structures in an image. A horizontal or vertical slit may be used to detect an axis feature. A well-known property of the two-dimensional Fourier transform is that the transform of a circularly symmetric object is also circularly symmetric. Thus, if one is trying to detect circular objects, an annular ring sampling geometry is appropriate. Peaks in this circular sampled signature would correspond to energy at a given radial distance in the transform space.

Template matching

Suppose that an image function of known form which is non-zero only over a finite rectangular region is perturbed by additive noise. The filter which maximizes the ratio of signal power to average noise power at the filter output at some spatial position is called a matched filter.[15] A detection decision may be made by sampling the matched filter output at the position and comparing this value to a threshold. A matched filter detection is not affected by a translation of the signal; however, it is sensitive to rotation. The matched filter is also sensitive to magnification changes. If only a small size variation in the desired signal is expected, then a search through several sizes may be reasonable.

Directional signatures

A simple but powerful technique for locating objects in fixed frame images consists of gray level directional signatures (e.g., Meyers, et al.).[16] A fixed frame image set consists of images of similar objects; for example, chest, head, or leg X-rays. Normalization is a significant but solvable problem. The directional signature consists of a function which is the sum of the gray level pixel values along a line normal to a reference line. The x and y direction signatures for a chest X-ray are shown in Figure 9. Note that major objects such as the clavicle, the lungs, the heart, and the diaphragm, may be located from the signatures. The signature information allows one to "zoom" in on a particular object. After an object has been located, measurements such as size or texture may be made.

Contour tracing

Edge structure is a significant feature in many radiographic images, and motivates the development of computer algorithms for extracting edge information. These programs are often called contour trace programs. A position in a picture where two objects meet will be characterized by a change in the gray shades in the picture array. This has suggested a search for points where the rates of change (deviations) of the picture array PIC (I, J) are large. Once these points are found, some form of logic is then used to connect these points together to form the boundary of the object. There are variations in these techniques including level slicing and contour tracing, but the results tend to remain much the same. This method has several problems. First, due to the noise that will invariably be present in the picture array, contour trace algorithm will tend to give gaps and false spurs. If there are touching and overlapping objects, a contour trace program will tend to switch from one object to another before a boundary of an object is completely traced. Contour trace programs are local algorithms and as such have the following problems: If mistakes are made, such as gaps, they must be corrected later. If there are several objects in the scene, a contour trace algorithm has no way of knowing that it is following a particular object and must be careful not to stray from it. Therefore this technique suffers from the problem of making mistakes which are passed on to the later stages of corrections, namely the pattern recognition stage. Thus, there is the very real problem of errors accumulating through the various processing stages.

Another method that has shown some utility is often called region enumeration. This method tends to be the inverse of contour tracing. In this method a point x = (i,j) in the picture array is located, perhaps by a raster scan, and one desires to identify the points that lie in the same region with x. There will be a property P(y) that one uses to determine if a point y is in the region. First, one finds x; then one examines the 8 neighbors of x. Let y be one of these points. If y has the property P, then y is placed in the region with x, otherwise it is not placed in the region. The process is repeated spreading out from x until all the contiguous points with the property P are placed in the same region. Often the property P will relate to the gray level of the point. For example, does point y have the same picture value as x? Or an average gray value of the region may be kept and y is tested to see if its picture value PIC(y) is close to the average value. Other P's can also be used. There are variations of these techniques in which one uses thresholding with region enumeration. Thus, we see that region enumeration seeks to identify all the points in a region, not the boundary points, and to do this it places points in the region if it is not on the edge of the object, while contour tracing looks for edge points.

The descriptive approach to image analysis

An approach to feature extraction that we usually refer to as the descriptive approach will now be briefly described. Other authors have described related work but have not applied it to radiographs.[17,18,19,20] It is possible that this method can be applied to a number of medical picture classes. Thus far we have applied the method to the analysis of AP chest X-ray with the goal of diagnosing abnormality in the heart and lungs. Since that time, the same program has been applied to AP X-ray of the knee and nuclear medicine images. There are several basic assumptions behind this method.

(1) The analysis of a scene should be top down. That is, one should first analyze the large objects in the scene at the lowest possible resolution and later analyze the finer objects in the scene as details of the large objects.
(2) Feature extraction and pattern recognition must be combined into a reinforcing system, i.e., a system with feedback. That is, feature extraction is not a separate process from pattern recognition; pattern recognition is an integral part of feature extraction and must be included from the beginning.
(3) One must have within the program a comprehensive description of the class of scenes to be analyzed. This description must guide the scene analysis system from beginning to end. The description should be in the form of data to a scene analysis supervisor so that one can readily analyze different scene classes without extensive reprogramming.
(4) Regions enumeration rather than some form of contour trace program should be used in primitive objective identification.
(5) All parameters in the program must be self-adjusting.
(6) The concept of field of vision is important in locating the true boundaries of objects. In many applications one must not only recognize the objects but must also determine the exact boundary of the object. An example is chest X-rays where one needs an accurate contour of the heart in order to diagnose heart disease. We believe that there are advantages in recognizing the fact that a scene is composed of objects that completely fill the area of a scene, i.e., every point in the picture is in some region and one always sees something even though it may be lumped into a catch-all category of background. These objects all fit together in jigsaw fashion to completely fill the space of the scene.

A graphical description seems natural in implementing assumptions 1 and 3. In particular a tree structure very naturally gives a top down description of the scene class. See Figures 10 and 11 for examples. In the program described in Reference 21 each node of the tree represents an object in the picture. The graph is data to the program. Thus when one changes picture

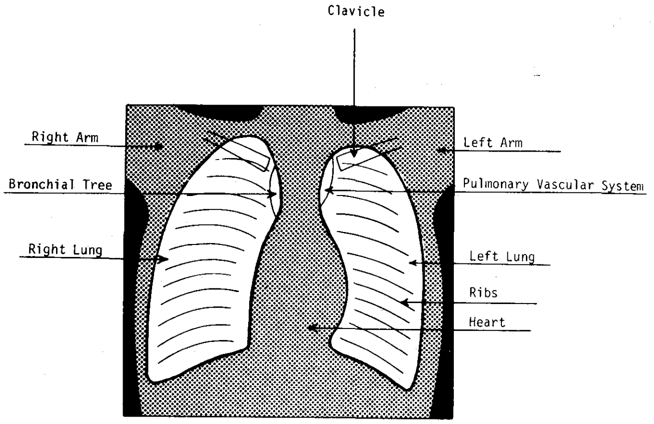


Figure 10—Anteroposterior view of chest

classes, one changes the data which describe the scene class. Attached to each node is a list of attributes for that object and a predicate that describes the objects relation to the other objects in the picture. The attributes might include such facts as:

(a) average gray level
(b) average number of points
(c) shape description such as higher order moments.

An important point to note is that using a top down search for the objects can be located with widely different resolutions. For example, our current version for the chest searches for the level one objects (diaphragm, mediastinum, right lung region, left lung region) at a 16 × 16 resolution.

One does not have to analyze the entire X-ray at once; one should first distinguish these objects from each other. If one desires very accurate descriptions of the borders, one can switch resolutions easily, retaining the information found at the lowest level. When our present program switches resolutions, it merely declassifies the boundary points between objects, then switches resolution and reenters the point classification program which decides to which object the point belongs. Thus, in moving to higher resolutions one merely reshapes the boundary to get an accurate description of the object.

The predicates attached to each node can be very valuable in identifying the objects. Many times an object in a picture has a border that has wide variations in the intensity of the picture function at the border. In places the border may not be visible at all. The predicates are a way to define the boundary of an object when the boundary may not be visible. For example, in Figure 11 consider node 7 and the predicate expressed in PL/1 notation

(L(7)|B(7)|(R(7) & (B(6)|L(6))))|
(U(7) & (B(4)|B(6)|L(6))))|
(L(7) & R(7)))|(U(7) & B(7))

This says that a point x can be put in a region associated with the left arm region only if one of the following conditions holds for the initial regions linked to the node which represents the left arm, Node 7.

(a) Region (7) is to the left of the point. This lets the region 7 grow without hindrance to the edge of the picture.
(b) Region 7 is below the point. Hence, the arm region can move toward the top of the picture without hindrance.
(c) Region 7 is to the right of the point and region 6 is either below or to the left of the point.

The other parts of the predicate are interpreted similarly.

Observe that the left arm region is thus controlled by the left lung region and points cannot be added to the region unless the left lung region lies to its left. Hence, the border is controlled by the predicate which reflects what the picture is of as well as the gray level characteristics of the picture. Figure 12 gives an example output for a program that we have implemented.

A program for the automatic computer diagnosis of rheumatic heart disease has been recently developed at the University of Missouri.[22]

The goal for this study was classification of the heart size from the standard PA chest radiograph into normal or abnormal category. If the cardiac size was judged abnormal, a further sub-classification was

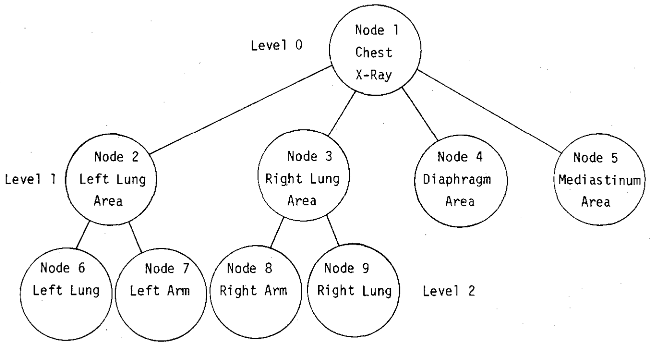


Figure 11—Partial vertical ordering of AP chest X-ray picture

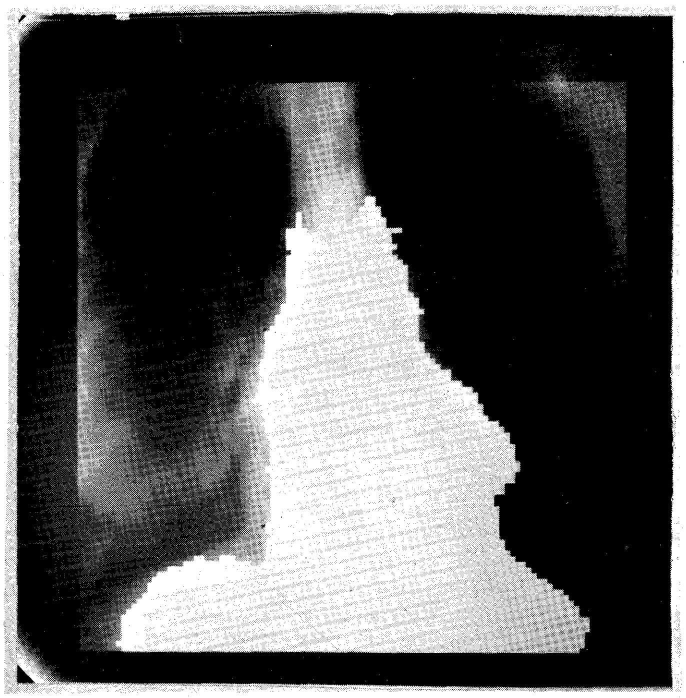


Figure 12a—Node 4—Diaphragm area

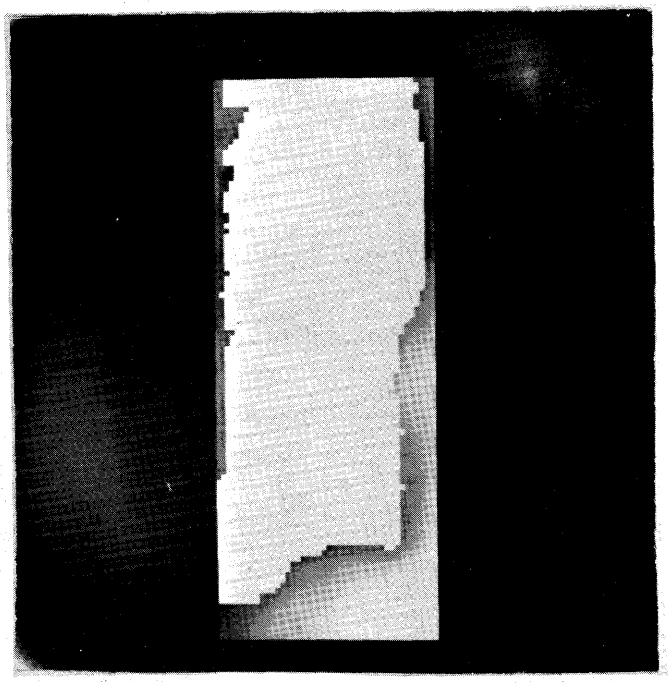


Figure 12c—Node 9—Right lung

accomplished into four separate categories of rheumatic valvular involvement. To test the basic normal-abnormal classification further, some non-rheumatic cardiac abnormalities in adults were also used as test cases. A test set of 279 cases was used. This set included 191 cases of rheumatic heart disease representing all combinations of rheumatic valvular involvement.

All patients in the abnormal group have had right and left heart catheterization and appropriate cinean-

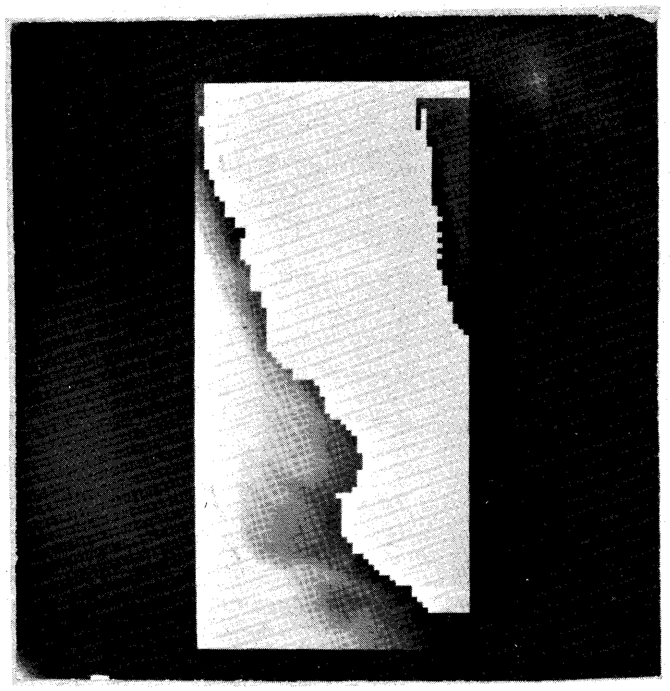


Figure 12b—Node 6—Left lung

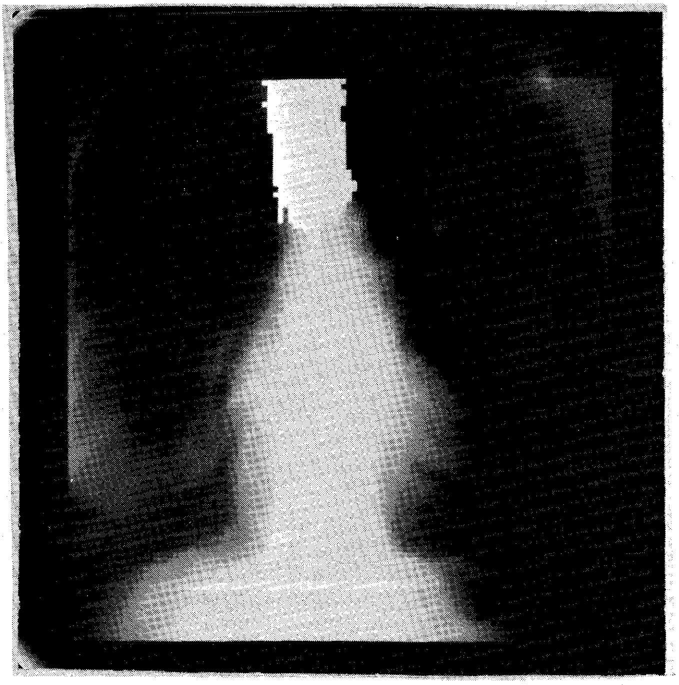


Figure 12d—Node 5—Mediastinum

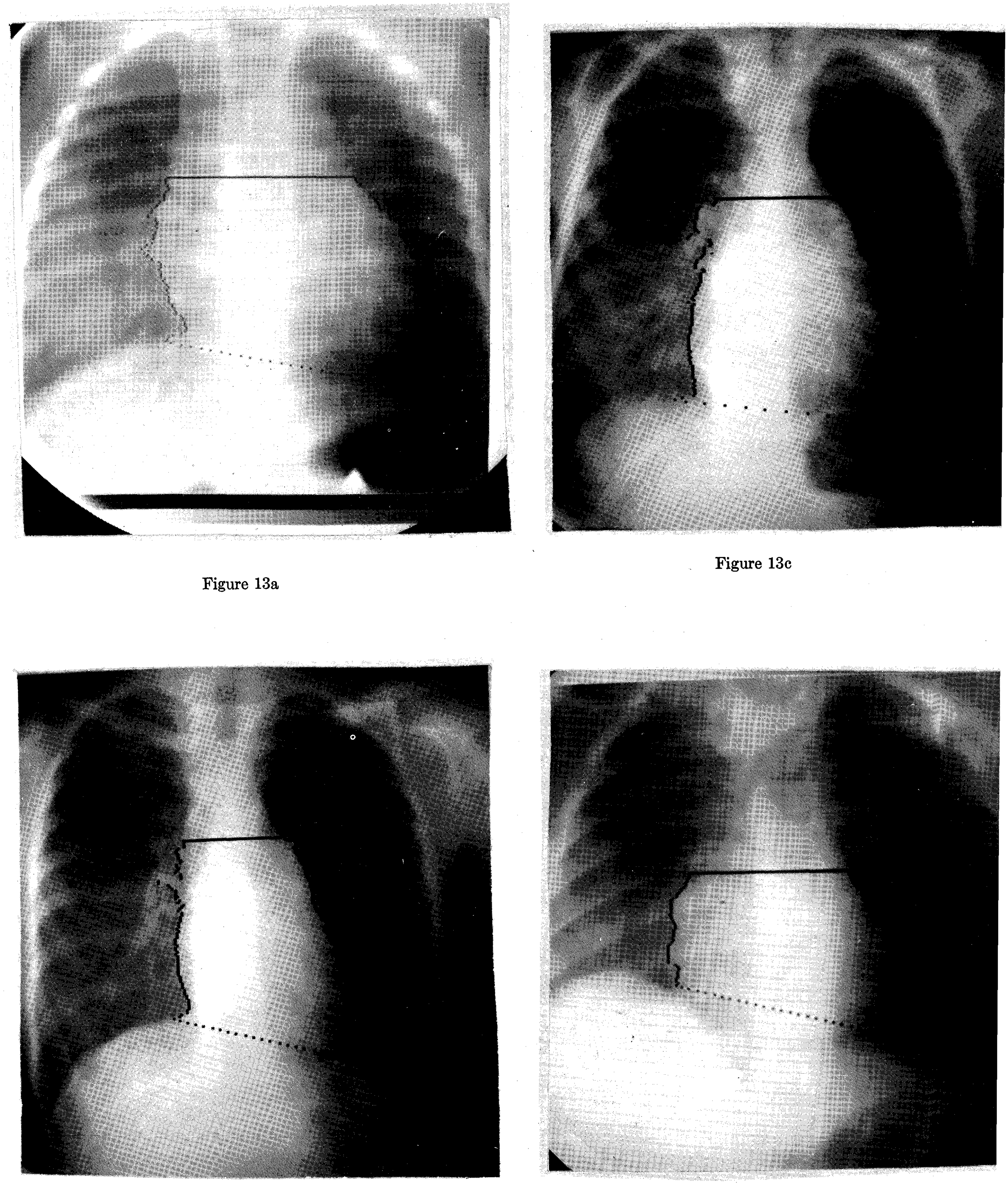


Figure 13a

Figure 13c

Figure 13b

Figure 13d

Figure 13—Cardiac contours outlined by computer

giography to establish the correct diagnosis; 88 normal chest examinations were also selected from patients with no history of cardiac disease, no murmurs, and no cardiac symptoms. These were obtained from routine psychiatric admission chest X-rays and preoperative and employee chest examinations. The normals were selected generally from the age group of 20 to 45 in an attempt to match the age group range of the rheumatic heart disease patients included in the set of abnormals.

The first 135 cases selected as the computer testing and training set were set aside as the radiologist test set library. This separate library includes normals and all categories of rheumatic heart disease, and comprises the basis of comparing the diagnostic accuracy of the computer and the radiologist.

The first step in the feature extraction process was to extract size, contour, and shape of the heart from the standard PA film. This was accomplished by an algorithm that constructs a cardiac rectangle around the heart. The cardiac rectangle is variable in size which is dependent upon the heart size. This cardiac rectangle is obtained by spatial signature analysis.

Once the cardiac rectangle was located, a technique for obtaining the closed cardiac surface was developed. The technique involved thresholding a gray level histogram to produce a one-bit (two level) representation of the chest image within the cardiac rectangle. The algorithm, using a gray level histogram, determines the threshold for converting the extracted cardiac rectangle into a two-bit representation (black and white) image of the heart.

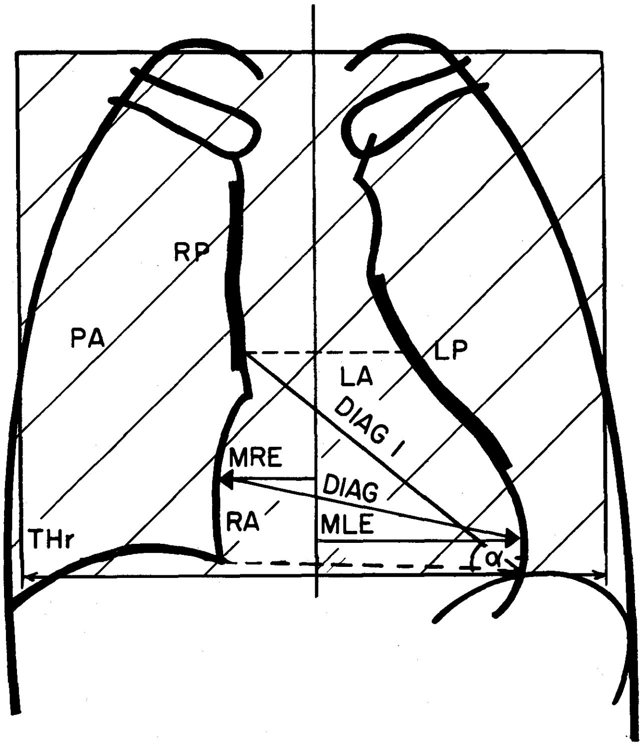


Figure 14—This shows the measurements and two polynomials extracted by the algorithm after the feature extraction phase has been accomplished

The next step was to outline the cardiac contour from this black and white representation. With the left and right outlines determined, the intersection of the right cardiac edge and the diaphragm was determined and a line was drawn from this point across to the intersection of the left cardiac edge with the diaphragm. A line was also drawn to connect the right and left top of heart (TOH) outlines so that the entire heart and portions of the pulmonary arteries are enclosed. A typical example is shown in Figure 13 a-d.

Area and extent measurements are now easily made from this closed contour. At this point, nearly all information from the standard PA chest radiograph needed for the diagnosis and classification of heart disease had been extracted, and the next step was measurement of the heart. The measurements taken are shown in Figure 14.

All cardiac measurements were normalized so that a ratio figure was obtained. The linear measurements were divided by Thr, the thoracic width, and the area measurements were divided by TA, the thoracic area. This allows correction for variation in heart sizes related to the patient's overall size. The same algorithm can also be used for different film input sizes, i.e., standard 14 × 17 inch, 35mm reduction, or chest photofluorograms.

Once these series of heart measurements have been extracted from the cardiac rectangle, this information was used as the basis for first classifying the case as normal or abnormal. If a particular case was abnormal, the classification went further by placing the case into the correct group of rheumatic heart disease.

The diagnoses were divided into 5 classes and the 16 possible combinations of aortic and mitral valve disease were divided into 4 separate groups. The classes considered by the classification scheme were:

Class 1:	Normal
Class 2:	Mitral Stenosis only
Class 3:	Other mitral valve lesions—MSMI; MI only
Class 4:	Aortic and mitral involvement
Class 5:	Aortic involvement—ASAI, AS only, AI only.

Computer classification was accomplished through the use of linear and quadratic discriminant function.[23] This classification method was selected because of its

			Testing	Training	
	Number in Class	Radiologist	Image Analysis Laboratory	Image Analysis Laboratory	BMD
Class 1, Normal	88	83%	88%	94%	94%
Class 2, MS	33	50%	21%	33%	33%
Class 3, MI, MS-MI	21	54%	10%	14%	29%
Class 4, Bivalvular	87	29%	72%	75%	70%
Class 5, Aortic	50	76%	48%	62%	60%
Total	279				
Overall Percent Correct Classification		62%	62%	69%	68%

Figure 15—Rheumatic heart disease classification results

relative simplicity and speed. Initially each part of the classification was processed using both the linear and quadratic discriminant functions. The selection for using a quadratic or linear discriminant function for each group in the final classification scheme was determined by the results: that discriminant function was chosen which gave the best results. If the accuracy was equal for both discriminant functions, the linear discriminant function was selected for the final classification scheme because it required less computer processing time.

In order to estimate the unbiased capabilities of the computer diagnostic classification scheme for the evaluation of new samples, the "jackknifing" test procedure was instituted. This procedure consists of first withdrawing ten percent of the cases from each of the five classes. The remaining ninety percent of the cases in each class with their measurement parameters and polynomials are then used to train the classification scheme. Each case in the withdrawn group is subsequently tested and classified into one of the five diagnostic classes. The withdrawn ten percent of the cases are then replaced into their respective classes, whereupon a different ten percent is withdrawn, tested, and replaced in an identical manner until all the cases have been tested. No case was tested more than once. This is a fair test, since in each case the algorithm had not "seen" the withdrawn test set until asked to make a diagnostic classification. A better test would probably have been to remove one case at a time and test each using the jackknifing procedure until all of the cases had been classified. However, this was not practical because of the large amount of computer time involved.

It would be difficult to judge the value of this automated feature extraction and classification algorithm unless its accuracy could be compared to the diagnostic accuracy of radiologists. For this reason, the following study was instituted.

Ten radiologists were asked to individually diagnose 135 representative cases of the cases evaluated by the computer. This group of radiologists consisted of 7 board certified academic radiologists and three third year residents whose training was nearly completed. Each radiologist was given the PA and lateral views and told that each case was either normal or rheumatic heart disease. He was asked to make a complete radiological diagnosis and record his answers on a form designed for the study. Each case diagnosed was counted as one physician's observation. Information on the forms was then transferred to punched cards and computer programs were written to separate the 639 physician observations collected. Not all of the physicians completed the task of reading all 135 films. A comparison of the physicians' results with the computer results is shown in Figure 15. The Image Analysis Laboratory results are those obtained using the pattern classification method just described. The BMD results are included for comparison.[24]

Overall accuracy was 62 percent for the computer and 62 percent for the group of radiologists. The overall accuracy was computed from the following equation:

$$\text{Overall accuracy} = \frac{\text{Total number correct diagnoses}}{\text{Total number of cases in test group}}$$

This example illustrates that although automatic computer diagnosis is quite different from the other methods, it is competitive and may out-perform them.

REFERENCES

1 L H GARLAND
Studies on the accuracy of diagnostic procedures
American Journal of Roentgenology and Radiation Therapy Vol 82 No 1 July 1959 pp 25-37

2 I T YOUNG
Automated leukocyte recognition
PhD Dissertation Department of Electrical Engineering MIT Cambridge Massachusetts June 1969

3 R S LEDLEY
Automated pattern recognition for clinical medicine
Proceedings of IEEE Vol 57 No 11 November 1969 pp 2017-2035

4 M MENDELSOHN J PREWITT
Analysis of cell images
Annals of the New York Academy of Sciences Vol 128 Article 3 1966 pp 1035-1053

5 J L LEHR R W PARKEY C A HARLOW L J GARROTTO G S LODWICK
Computer algorithms for the detection of brain scintigram abnormalities
Radiology Vol 97 No 2 November 1970 pp 269-276

6 G NAGY
State of the art in pattern recognition
Proceedings IEEE Volume 56 May 1968 pp 836-864

7 DOI KUNIO
Optical transfer function of the focal spot of X-ray tubes
American Journal Radiation Therapy and Nuclear Medicine October 1965

8 R H MORGAN
An analysis of the physical factors controlling the diagnostic quality of roentgen images
American Journal Radiation Therapy and Nuclear Medicine December 1965

9 R P KRUGER S J DWYER E C MAHEN
S J DWYER A J CARLSON G S LODWICK
Image analysis of radiographs
SWIEECO Record of Technical Papers Dallas Texas April 1970 pp 147-149

10 R SELZER
Improving biomedical image quality with computers
JPL Technical Report 32-1336 October 1968

11 A ROSENFELD
Picture processing by computer
Academic Press New York 1969

12 E L HALL S J DWYER
Frequency domain design of spatial digital filters
IEEE Information Theory Conference June 1970—Also University of Missouri-Columbia Image Analysis Laboratory Technical Report 101 June 1970

13 T G STOCKHAM et al
Nonlinear filtering of multiplied and convolved signals
IEEE Proceedings August 1968 pp 1264-1291

14 G G LENDARIS G L STANLEY
Diffraction-pattern sampling for automatic pattern recognition
Proceedings IEEE Volume 58 February 1970 pp 198-216

15 H C ANDREWS
Automated interpretation and classification of images by use of the fourier domain
Automatic Interpretation and Classification of Images Academic Press New York 1969 pp 187-198

16 P H MEYERS C M NICE H C BECKER
N J NETTLETON J W SWEENEY
G R MECKSTROTH
Automated computer analysis of radiographic images
Radiology Volume 83 December 1964 pp 1029-1034

17 PREPARATA FRANCO SYLVIAN RAY
An approach to artificial non-symbolic cognition
UILC-ENG 70-223 University of Illinois Urbana Illinois July 1970

18 I MACLEOD
On finding structure in pictures
Picture Language Machines S Kaneff ed Academic Press Inc London-Ltd 1970

19 H G BARROW R J POPPLESTONE
Relational descriptions in picture processing
Machine Intelligence 6 American Elsevier Publishing Company 1971

20 C R BRICE C L FENNEMA
Scene analysis using regions
Technical Note 17 Stanford Research Institute April 1970

21 C A HARLOW J L OTTO D L HALL
G S LODWICK
Feature extraction in images
Technical Report Image Analysis Laboratory Electrical Engineering and Radiology Departments University of Missouri-Columbia Columbia Missouri July 1971

22 R P KRUGER D L HALL G S LODWICK
S J DWYER III
Computer-aided diagnosis of radiographic cardiac size and shape descriptors
Technical Report University of Missouri March 1971

23 J R TOWNES
The Gaussian mixture decomposition of estimated probability density functions
PhD dissertation University of Missouri-Columbia August 1971

24 W J DIXON ed
BMD Biomedical computer programs
Second edition University of California Press Berkeley and Los Angeles 1968

The impact of computing on the teaching of mathematics

by WALTER KOETKE

Lexington High School
Lexington, Massachussetts

A discussion of the impact of computing on the teaching of secondary school mathematics must begin with an examination of how modern computing facilities have entered and are entering the schools. The manner in which facilities are obtained often influences the way in which a mathematics program is implemented.

Most secondary schools with computing facilities used by the mathematics department have acquired their facilities in one of two ways. The key to both methods, however, is a motivated, aggressive member of the mathematics department. The reasons for this motivation vary dramatically, but without such an "inside agitator" many schools would yet be without computing facilities.

The first of the two ways by which facilities are initially acquired centers around the manner in which the school conducts its administrative data processing. Several service bureaus that specialize in educational applications have agreed to run a limited number of student programs with very little or no cost to the school. Local industries have been remarkably cooperative in providing limited computer time for the running of student programs. Many schools have had their own computing facilities used for administrative processing for the past 10 to 12 years. These facilities are usually capable of running student programs in a batch processing mode. The second way of acquiring computing facilities for mathematics requires that the school budget additional funds. Although these facilities are more desirable from an educational point of view, the add on cost makes them understandably harder to initiate. Such facilities include purchasing time from either a large commercial time sharing system or from one of several smaller time sharing services dedicated exclusively to serving the educational community. In fact, several secondary schools and small colleges have obtained small time-sharing computers that not only serve their own needs, but also permit them to sell computer time to nearby shcools—usually at a very reasonable rate. Mini-computers supporting multi-user, one language systems are being purchased at an increasing rate. Mini-computer systems that support from one to seven users are now being marketed not only by the major computer corporations, but by many smaller companies who claim to tailor the software and hardware to suit the needs of education. Several batch processing minis are also available with such options as optical card readers that purport to provide the speed of batch processing without the inconvenience of key punching all input.

Having outlined the manner in which computing facilities are obtained, a discussion of the way in which the mathematics department uses these facilities is appropriate. During the past six to eight years many schools have experienced an evolutionary sequence of four distinct applications. Unfortunately, most school systems with newly acquired facilities seem to be beginning the same four step process—without deriving much benefit from the prior experiences of others.

The first of the four steps is to teach programming for its own sake. The main objective of a course is to "learn to program." No specific attempt is made to explore mathematical topics not required to understand the programming language. This first step was a very natural one several years ago when batch processing FORTRAN was the most widely available language in schools, but this step should no longer be necessary with the more user-oriented languages and facilities available today.

The second of the four steps is to teach a course most appropriately titled "computer science." The tcpics of this course usually include a programming language, an introduction to the "insides" of a computer and how it works, and some computer related mathematics. This course is almost always offered as an elective for capable, interested students. The depth of this course is usually directly related to the computer related experience of the teacher. Several acceptable textbooks for secondary school students have been recently published for "computer science" courses.

The third step is that of using the computer as a supplement to the existing mathematics program. Occasional computer related assignments are given that complement the "traditional" mathematics being studied. Students generally have not volunteered for a special elective, but have simply enrolled in a mathematics course that happens to use the computer. This application is often thrust upon a teacher who is only half convinced of the computer's usefulness. This leads to the puzzling dilemma of the teacher's excusing an occasional student who "doesn't like the computer," but the many students who "don't like word problems, or factoring, or solving a series of three or more equations, or . . ." are told that the choice of curriculum is not theirs. The creative teacher, however, has found this supplementary work to provide an excellent opportunity to experiment with many different ideas. The results of such experimentation is one of the factors leading to the now emerging fourth step of this evolutionary process.

The fourth step is that of using the computing facilities as an integral part of the mathematics curriculum. Computer use does not occur only as a supplementary topic or in an elective course, but is an accepted part of the regular mathematics program. This step is just beginning to be achieved in a few school systems. Although this step is the goal of many mathematics departments, supporting textbooks are not yet widely available. This use will, however, develop as the major application in most secondary schools. In this step the computer is used as one of several tools available to the student of mathematics. The computer is used when its application is appropriate and ignored when it is not.

Let's look more closely at how the computer is used as a problem solving tool in this fourth step. The problem solving application is not a drill and practice program used to review rote facts, it is not a computer-assisted instruction application in which the computer presents a pre-programmed lesson, and it is not an application of computer-managed instruction in which the computer is used to log the progress of individual students. Rather, the student is given a mathematics "assignment" just as he is given an assignment if no computing facilities are available. The difference, however, is that the solution to some of the problems may require the use of the computer. In many instances, the student himself must select the problems appropriate for computer solution. When computer use is required, the student must write his own program, enter it into the computer, and complete all required debugging until a solution is obtained. The only "canned program" the student uses is the programming language itself which is the vehicle utilized to solve the mathematical problems. Students in such a course are likely to average writing one or more programs per week. They are, of course, free to write more, but one per week is the number they are required to complete. This type of application is best implemented using some type of interactive terminal. Although batch processing is an acceptable alternative, some important benefits are lost without on-line interaction.

Implementation of the problem solving application seems most successful if terminals are available in a mathematical laboratory rather than a classroom situation. Since a single interactive terminal is most effective when used by one or possibly two students, classroom use is not very efficient unless the terminal is being used as part of a demonstration in connection with some type of projection device. Although some schools have unfortunately experienced occasional incidents of vandalism, an open laboratory in which students are free to schedule their own use of the computer has proved very successful in many installations. Although some type of control is necessary so that many different students can use the facilities, this control should be only that which is absolutely necessary. Ideally the only enforced regulation is one that limits the total time a student is permitted to use the facility. Such a limit not only increases the number of students who can be served, but also forces the student to be "better prepared" when he does use the computer.

How do students using the computer as part of the regular mathematics program learn to program? They should be able to learn the required programming as part of the mathematics course. The programming language used must, therefore, permit the implementation of sophisticated algorithms yet also be very easy for the beginner.

The past 12 to 18 months have reflected school systems acquiring computer facilities at a continuously increasing rate. I suggest that this rate will continue to increase until it "soars" in another 12 to 24 months. There are several reasons for the presently increasing growth rate and the projected increase in this rate. The importance of these reasons varies among school systems, so the order in which these reasons are listed in this paper is not significant.

First is the simple fact that the cost of acquiring computer facilities has been reduced to an acceptable level. More than one school committee has noted that computer facilities can now be acquired for less than the average salary of a teacher, and that's not very much. The reduced cost has been evidenced in the purchase price of mini-computers, the rental of time-shared terminals, and leasing agreements of many batch processing systems. If there is a single most important reason for the increased growth rate, the reduction in price is the most likely suggestion.

A second reason is an increased awareness by both teachers and administrators that computing facilities have become a necessary part of the mathematics program. Although controlled experiments with significant quantitative results demonstrating the value of using the computer are almost non-existent, overwhelming qualitative evaluations ardently support the acquisition of computer facilities. In one of the few quantitative studies seen by the author, students using computer facilities and a control group were given a pre- and post-test in abstract reasoning. The increase in the group mean of the group using the computer was four times that of the control group.

The wider availability of BASIC is the third reason for the increasing growth rate. BASIC is not here defended as the "best" programming language. It is, however, offered as the most appropriate language for present secondary school requirements. Remember that the primary application of the computing facilities is problem solving, not computer science. The computer is to be used as a problem solving tool, thus the programming language itself should not be a problem for the students. Languages such as FORTRAN and APL are themselves problems for most students. The language used must also be well documented, not just in vendor's operating manuals but in the wider literature of mathematics. Languages like FOCAL, CAL and TELCOMP are certainly easy to learn, but they are not often referenced in the literature. BASIC, however, satisfies both of these requirements—it is very easy to learn and commonly referenced in the literature.

A fourth reason for the increasing growth rate is the also increasing number of teachers who want to and are qualified to utilize computing facilities with their mathematics classes. This is partially due to the growing number of good inservice courses being offered in many schools of education. Many new inservice courses are, however, strictly "how to program" classes that do not help teachers implement their new skill in the classroom.

Local school board pressure to "keep up with the competition" who have already acquired computer facilities is the fifth reason. Although this is not a very defensible reason for action, the movement is in the direction of improving education so we should accept rather than condemn it. Such action, however, occasionally results in disastrous experiences when computing facilities are thrust upon unprepared or uninterested teachers.

The sixth and final reason to be listed has not yet happened, but will soon be responsible for giving additional impetus to the growth rate. The fourth and most desirable application of computing facilities was as a tool in the regular curriculum courses. Although this is now being done in some school systems, it is done without the benefit of a single textbook for the students. Such textbooks are, however, now being prepared by several major publishers. When these textbooks are actually marketed in another 12 to 18 months, large numbers of already capable teachers will have the materials for which they've been waiting. This should result in an even greater demand for computing facilities in secondary schools—including those schools that already have modest facilities that are not yet required to serve the majority of mathematics students.

Having examined the manner in which secondary schools acquire computing facilities, the way in which these facilities are used, and the reasons for the continually increasing interest in acquiring such facilities, we will now partially answer the most important question—"Why use the computer in the study of mathematics?"

The history of mathematics shows that the major emphasis of mathematics has shifted from geometry, to algebra, to calculus, to non-Euclidian geometry, and now to algorithms and computation. As the emphasis of mathematics shifts to best serve the needs of society, so also should mathematics education. Topics related to algorithms and computation can no longer be treated as supplementary; they have become a very necessary and important part of the mathematics curriculum.

The history of mathematics also reveals that mathematicians have spent a great deal of time creating ingenious ways to avoid computation. Modern computing facilities have, however, eliminated the need for many of these clever computational aids. Once difficult topics have become quite easy, and many new topics have become very important. Quite simply, the computer has rearranged the priorities of modern mathematics and these new priorities should be reflected in the mathematics classroom. Consider, for example, the "traditional" student of first year algebra and his accrued ability to find the zeroes of a function. He can probably find the zero of any linear function he encounters. If he was in a fast moving class he can even solve quadratic functions—especially those that are factorable. Can he solve other polynomial functions of higher degree? No. Can he solve any other type of function? No—he may not even know that other types of functions exist. If the student is persistent for an additional two to three years, he will eventually learn Newton's method or a similar algorithm that can be used to find the zero of almost any function. The path to this general algorithm is littered with "useful" techniques such as factoring a quartic or even higher degree polynomial to obtain the zeroes. The student in a first year algebra class that has access to computer facilities can study functions and their zeroes in a very different manner. For example, he can locate zeroes of almost any function by searching the domain for a change in either the sign or the slope of the function. This technique will not only reveal the zeroes, but also permit the student to explore maximum and minimum values of a function in an informal yet precise manner. There are of course, many similar examples of now important topics that can only be taught very superficially if at all without access to computing facilities. Listing these topics is not, however, the purpose of this paper. Topics that fall under the general heading of Monte Carlo Methods are fascinating to almost all students. Computing facilities permit the inclusion of some of these now important topics. Probability and statistics cannot only receive increased emphasis with computer use, but realistic problems can be approached rather than superficial models that don't really communicate the meaning of or need for the ideas being studied.

In addition to reflecting some of the priorities of today's mathematics, use of the computer as a problem solving tool helps teach that which is the real purpose of all education—how to think. Although teaching students how to think is the goal of education, there is no course at any grade level in which that is the prime topic, nor is there any methods course in related teaching techniques offered to prospective teachers. Computer facilities will not immediately change this dilemma, but the analytic and algorithmic thought processes are so much a part of programming and related problem solving that students cannot avoid learning them in a well presented course. As one would hope, there is a great deal of qualitative evidence indicating that students do transfer these processes to other disciplines as well. Fortunately, a growing number of mathematics educators have begun to realize that a student's mathematics education should be judged by how he reacts to problems rather than by a list of courses and grades. Computer assisted problem solving and accompanying algorithmic thought help students develop a very sound approach to many types of problem solving in many different disciplines.

Some of the ways in which computer use helps develop analytic and algorithmic thought processes might be clarified by an example. Consider the problem of describing the solution set of a quadratic inequality. A traditional text presents this topic by first listing several specific examples, then asking the student to solve several other specific examples. In a subsequent class, the student is told which of his answers are wrong. A presentation of the same topic with computing facilities available might well begin in the same manner—but it would go further. Students would first be asked to solve a few specific examples using pencil and paper, then asked to write a program that would permit a user to enter any set of values for A, B and C, then print the solution set of the inequality $Ax^2+Bx+C>0$. To write the algorithm for such a program the student would likely have to create and solve many specific examples. He would not bother with the arithmetic exercise of solving examples he knows how to complete. Instead, he could concentrate exclusively on the cases for which he is unsure of the solution set. When testing this program, as with many other programs, the student must "debug" his algorithm. He is not faced with a simple right or wrong situation, but one in which he is likely to be partially right. He must then determine which portion is not correct and the required procedure for correcting it. The entire process of debugging a partially correct algorithm is a realistic valuable experience that is rarely encountered in more traditional courses. Few problems the student will face in any areas of his life will be countered with a solution that is all right or all wrong. The solution will be partially correct and will require repeated debugging.

As computing facilities become available in the large majority of secondary schools, the students, the teachers, and the computing industry itself will feel the effects. The remainder of this paper is devoted to an examination of these effects. Note that some of these effects are already a reality while others are the predictions of the author.

The most immediate and obvious effect upon students is their high degree of motivation. The reason for this motivation changes from student to student. In fact most students seem to sustain their motivation for a shifting set of personal reasons. Skeptics point out that use of computing facilities in a mathematics class does not motivate all students. That is certainly true, but it does motivate most of them. Unfortunately, the same cannot be said for either the content or presentation of most traditional classes. Skeptics also profess that most of the student motivation is simply fascination with a new type of hardware. Although this might well be true at first, the source of student motivation soon shifts to many diverse, more personal reasons. I suggest, however, that even if new hardware were the one and only motivating factor, that's fine. Most teachers are well aware of the fact that motivating students is at least half of their job, and they would be delighted to find any proven method or "machine" that will develop sustained motivation in their classes.

Another effect upon students is that the present "ability grouping" of classes will have to be restructured. Since these groupings often represent present motivation or the results of past motivation, any factor that significantly alters motivation will also alter the groupings. Preliminary evaluations of computer use in mathematics classes indicate that the gap between the "good" and "bad" students is significantly reduced. There are many possible reasons for this. For example, the "good" students are already highly motivated so they do not benefit from that aspect of using the computing facilities.

The study of algorithms and computation permits students to make genuine discoveries. They are able to develop significant new computing algorithms and improve upon those that already exist. Students cannot improve upon the proof of a theorem in plane geometry nor uncover an unknown property of the real numbers. These are unlikely occurrences at any level of mathematical sophistication, and those suggesting that secondary students often make such discoveries kid only themselves. Although a skillful teacher could lead some students to the belief that they have made a discovery—and indeed they often do make personal ones—the discoveries themselves are only contrived and the results are already well-known. In the area of algorithms and modern computation, however, the secondary student is participating in a discipline that is about 20 rather than 2,000 years old. Many students soon learn that they can indeed uncover. flaws in existing algorithms, as well as create new and better ones.

Computing facilities provide many students with a means for exploring previously unexecutable ideas. This benefit of computer use has evidenced itself not only in mathematics but in several other disciplines as well. A favorite student question, "What if . . .?" must go unanswered fewer times when computing facilities are available. More significant than the answer to this question is the fact that the student has the means to answer it himself. In 1970 a high school student discovered the 21st, 22nd, and 23rd perfect numbers. (Perfect numbers are those positive integers that equal the sum of all their divisors excluding the number itself—such as 6 and 28.) Was this student "above average?" Sure he was—so were Pythagoras, Euclid, Fermat, Descartes, Euler, and many other brilliant mathematicians who unsuccessfully sought perfect numbers during the last 2000 years. Is this student the greatest mathematician the world has known? Probably not! He was, however, a creative high school student with access to computing facilities. I feel that the three numbers he discovered and the theorems he created are of less importance than the simple fact that the entire refining process of idea to theory to algorithm to execution to proof to announced results was carried out by a high school student with access to computing facilities. I contend further that such significant work will become an accepted if not expected product of secondary school mathematics during the next decade.

Computing facilities have made and will continue to make secondary school mathematics more relevant and more interesting for students. Pinpointing a single reason for increased interest and relevance cannot be done as these vary widely between students. The simple fact that execution of a program provides the student with immediate reinforcement does much to heighten and sustain his interest. He quickly learns the extent of his success, and if he is well prepared with appropriate test cases, he also receives an immediate, impartial evaluation of his algorithm. Interest is also supported by the student's ability to solve a problem in his own way. No longer will he be told he is wrong because the first few steps of his procedure are not what his teacher expected to find. The student can now complete the entire procedure in his own way, then immediately confirm the validity of his work.

Students find computer related mathematics relevant because they are using what they know is modern technology. They are also able to solve much more complicated problems than the usual oversimplified tasks they are given. In our society of space flight, push button wars, ecological crises, widespread famine, and widespread excess—we cannot afford the luxury of oversimplification. We must be as precise as possible, and where approximation occurs it must be clearly identified and its implications understood. Even the simple concept of percent error is foreign to many traditional mathematics classrooms, yet students want to know and should know about approximation. This is yet another example of the fact that a solution to a real problem is rarely all right or all wrong. An almost right solution, however, may not be of much value unless the "amount of wrongness" can be clearly evaluated. The relevancy of the activity in mathematics classes is also heightened by the fact that students can utilize the computer in other disciplines. Although there is often little application of mathematics when students create simulations for use in science or history, tutorial programs for use in English or foreign language classes, or any of a host of other applications, there is an increased interest in both disciplines. Although the direct application of mathematics may be slight, the application of the algorithmic thought process and the application of problem solving techniques are indeed interdisciplinary. Such student projects are not only very valuable to the student, but quite often the teachers will also become involved in a meaningful interdisciplinary study.

Having examined some of the ways that computing facilities have affected and will affect students, let's now examine the corresponding effects upon teachers. The role of the teacher is altered in several important ways. Most apparent is that the time spent reminding students that they must pursue other studies in addition to mathematics far exceeds the time spent in pursuit of students who are not attempting to complete their work. Quite simply, teachers spend more time convincing students to go home than they do coercing them to do the assigned work.

The teacher's role will become much more that of an advisor rather than that of a lecturer. This new role will place much greater demands upon the teacher, and may be one reason that some teachers have resisted using available computing facilities. The role of advisor means that the teacher must maintain a working knowledge of more mathematics than he is now required to do. Many secondary school mathematics teachers have taught only a single subject—geometry or first year algebra or . . . —for several years. Even though this restricted teaching load may continue, students' pursuit of their own interests and many incidental student questions resulting from assigned programs will not be so restricted. The teacher's knowledge will be inadequate unless it extends well into many areas of mathematics.

Can teachers be expected to know all of the answers? Certainly not, yet this response is not a source of comfort but of discomfort for many teachers. Students using computing facilities remove the security of the teacher who, with his answer key, does indeed have an answer to almost all questions. This artificial security will be undermined even further because of the newness of many computing algorithms. Few teachers believe they are smarter than all of their students—but they have the decided edge of experience. However, when computing facilities are introduced, the edge of experience is removed. There are clearly many teachers who are very uncomfortable in this unfamiliar situation.

A teacher's mathematical background is likely to require some refreshment. One of the reasons for adoption of computing facilities is to reflect the changing emphasis of today's mathematics. The experienced teacher may well find himself in need of several inservice courses to update his own background in mathematics. Topics such as Monte Carlo Methods, Probability and Statistics, Approximations and Error Analysis, as well as Programming itself are some of those that may require additional study. Certainly an extensive, nationwide effort to renew support for inexpensive, convenient inservice education is in order. The positive effects of such an effort were clearly evidenced when similar support was given to the various SMSG mathematics programs. Although an effort of that great a magnitude may not be required, the majority of mathematics teachers do require some type of inservice education and that is no small undertaking. The benefits of these courses should, however, far exceed the possible inconvenience and certain expense of conducting them.

The teacher must also become familiar with a new structure in the courses he is teaching. Of course programming is new and the topics themselves will be somewhat changed and rearranged. Teachers will be able to concentrate on problem solving rather than arithmetic. Any experienced teacher can relate many stories of students becoming so "hung-up" in the arithmetic of a problem that they lose track of what the problem is all about. A traditional course in first year algebra examines polynomials that can be factored as the product of two binomials. The next topic is likely to be algebraic fractions and various techniques for their manipulation. Unfortunately, much of the topic of algebraic fractions is likely to be completely obscured because students approach almost all problems as additional exercises in factoring. Adoption of computing facilities reduces the likelihood of students getting "hung up" on the arithmetic of a problem. One can't resist the observation that problems will no longer have to have answers of 1, 10 or 100 so that the arithmetic can be more readily done by hand.

The teacher will also shift the emphasis of a mathematics course somewhat away from the current stress on structure. The structure of mathematics is indeed important, but that does not mean that the best way to teach the subject is to proceed in an orderly step by step path through this structure. After all, this certainly wasn't the order in which the mathematics was created. The structure will not, of course, be totally ignored, but it will not be emphasized until more advanced courses.

Teachers will also find themselves in a delightful new situation—they will be able to get genuinely excited about the work of their students. Students in mathematics classes using computing facilities can indeed create an exciting algorithm just as an art student can create an exciting sculpture. Clever teachers have appeared enthusiastic when a student solves a problem correctly, just as they have generated a sincere enthusiasm when a particularly "slow" student finally learns a certain technique. They have not, however, often experienced the exhilarating feeling of sharing a student's enthusiasm for something he alone has created. The strongest advocate of an idea is the person who creates it. As the computer permits the student and teacher to share in the development of new ideas, a completely new and very desirable student-teacher relationship will emerge.

Finally, let's examine some of the effects that secondary school computing facilities might have upon the computing industry. Clearly secondary schools represent a new and growing market and thus a new and growing source of profit. The nature of this market, however, is different from most others. The most important single factor in education is cost. Contrary to your feelings as taxpayers, schools do attempt to save money whenever possible. The continuing emphasis on minimizing cost permeates all phases of computer related activities. Although this concern for economy is often very desirable, it often becomes dictatorial. Many competent and valid studies of needed computing facilities have been prepared. The educational pros and cons of several systems are evaluated and closely scrutinized by several diverse committees. And then the decision is made—the school system accepts the least expensive of the proposals.

If a school system adopts computing facilities and has a bad experience with them, this experience is communicated to many, many other schools. Vendors should, therefore, make a special effort to see that the staff of a school is well prepared to utilize the facilities they are marketing. The educational customers of today are surprisingly unsophisticated in comparison to their counterparts in industry. They require a larger than usual amount of "hand-holding" both before and after a sale, and require very solid software and hardware. Although the situation will eventually change, most schools can today only use the software provided by the manufacturer. They do not have the internal capability of tailoring that software to better suit their individual needs.

Other characteristics of the educational market include the fact that the "state of the art" is not an important consideration when hardware is selected. The very latest equipment is a luxury that very few school systems can afford and even fewer systems require. Reliability and durability are far more important than extra speed or exotic features. Most school systems will acquire their computing facilities gradually over a period of several years. Thus vendors must provide very flexible systems that can be expanded with little or no "wasted" expense, yet remain an effective system at each stage of expansion. As has been evidenced previously in this paper, schools are expected to increase their demand for interactive terminals on which students can implement their BASIC programs. The interactive requirement may be altered in some situations, but it is unlikely that BASIC will be replaced during the next few years.

Finally, I hope that one effect upon industry is that more of you get involved in secondary schools. I intend no disrespect for my profession when I state that we need all the help we can get. Offer to teach a special course in your local school system to students, to teachers or both. Assist your school administration and school committee in evaluating various facilities and the benefits they offer. The educational process you support now will repay all of us in the future—and while providing this support you too may have the opportunity to share the real joy of working with today's young people.

Computing in the high school—Past, present and future—And its unreasonable effectiveness in the teaching of mathematics

by WARREN STENBERG

University of Minnesota
Minneapolis, Minnesota

Five years ago computing was trickling into the secondary school scene, but today it's a torrent. Various hardware and software developments over the last two decades have made the computer ever more appropriate for this environment. These developments were: the shared program; the replacement of vacuum tubes by cores; compilers and procedural languages and finally—time sharing. The latest development has been the great reduction in the price of time sharing over the last year.

Six or seven years ago the pattern of computer use in the schools was very different from today. Some schools purchased table top computers with memory too small to contain a compiler so that students programmed in assembler language. (This had some disadvantages which I'll comment on later.) A second pattern saw the teacher carrying the student's program decks to a friendly college or industry to be run in batch mode during off hours. This method had the disadvantage that it took several days to get the three or four runs usually necessary for debugging.

But today we have an entirely different scene. Today it's all time sharing. I illustrate by describing the situation in Minnesota where I have my best access to information. For the last three years we have had time sharing in a few schools in the Twin Cities area on an individual port rental basis. Today every Junior and Senior High School in Minneapolis and St. Paul has at least one console. This also holds true of most of the suburbs. In the outlying areas of the state, time sharing has not as yet made a very strong showing.

After this glimpse at the hardware trends in the schools, it's time to see what the computer is doing there. The types of computer use fall generally into four categories:

1st Computer Assisted Instruction (CAI)
2nd Student use of canned programs including simulation
3rd Student programming
4th Computing courses

(The first three categories involve use of the computer in courses in the traditional curriculum.)

CAI is essentially a programmed textbook with the computer turning the pages. My personal prejudice (shared by many people I have talked to) is to be very skeptical of CAI as a substitute for the teacher in the classroom. But I see a great future for it in the areas of drill, review and remedial work. So far CAI is nonexistent in Minnesota secondary schools. At present the main deterrent to CAI is cost. If the cost factor were eliminated there would still be the impediment of the scarcity of CAI materials. At the University of Minnesota, CAI is extensively used in the beginning German course. The cost per student hour at the console is about one dollar; each student spends about five hours per week at the console. This totals up to $50 per student per 10 week quarter. In addition the hardware (CRT terminals) is exorbitantly expensive. Also the development of the software for this course was about $20,000 not counting the overhead of maintenance of the CAI center at the University of Minnesota. This program would have been impossible without generous foundation support. According to experts of the University of Minnesota, economically feasible CAI depends on two developments: first dramatic hardware improvements—probably replacement of cores by integrated circuits; second, increasing the amount of activity at the terminal end of the system—computer controlled slide projectors, film clips and audio cassette players. Even assuming that the necessary hardware developments will be forthcoming, the development of the course materials will require national dedication to the project and many millions of dollars of financial support. Even in this era of electrifying change, a decade would seem a conservative estimate for the realization of the potential of CAI.

Computer science courses have not as yet played a major role in the computer use in the secondary scene but they now seem to be coming up fast. In Minneapolis three high schools currently have such courses. Next year six schools will have such courses and in two years it is estimated that all the city's high schools will have them. Representatives of the University of Minnesota computer science department believe that within a few years the content of their present first course will be so widely offered in high schools that it will no longer be offered at the University except as a remedial no credit course.

But we are confronted with a very definite problem in connection with this movement of shoving the first computer science course back into the high school. The problem is that no standard curriculum has as yet been developed for these courses. Each teacher is "doing his own thing." A great deal of highly creative course development is taking place with bits and pieces being assembled from various sources; textbooks are generally not used. This is actually a very healthy development but still it creates a problem for the colleges. For they cannot know just what the students passing through these courses know about computing. So there is an urgent need for state departments of education to establish guide-lines or syllabi for computing courses. These should be worked out in collaboration with high school teachers and college computer science departments. These guide-lines should be sufficiently flexible as to encourage the creativity and originality of the individual teachers but they are altogether necessary lest the high school programs go to waste. First steps in this direction are now being taken in Minnesota.

Actually we can foresee three kinds of computing courses in the high school:

(1) *The College Preparatory Course.* This emphasizes algorithms and programming probably with the major stress on mathematical algorithms. It includes subscripted variables, functions and procedures. This course could be an alternative for one semester of mathematics.

(2) *The Engineering Course.* This course would include some algorithms, but the stress would be placed on computer logic and circuitry, binary arithmetic etc. In a lab associated with the course students would construct models of computer components. The course might include a quick pass over some elementary principles of electronics. The course could be used as a substitute for a semester of science.

(3) *The Vocational Course.* This course would consist of data processing techniques, the COBOL language, and an experience with operating a computer. Though primarily conceived for the non-college bound student, the course could be useful for the potential student of business.

The latter two courses are still in the category of pipe dreams since the teaching personnel just does not exist.

So far we have been discussing computing courses in the secondary schools. It's time that we should, turn our attention to the other face of the coin—the use of the computer in the conventional curriculum. This category makes up the great bulk of computer use in the schools at the present time. It is first necessary to recognize the sharp dichotomy between the use of the computer in mathematics courses and the use in other curricular areas. In mathematics courses virtually all computer applications involve the student in constructing his own algorithms and writing his own programs. Outside mathematics the situation is completely reversed. Here the student does no programming himself but works with canned programs where he supplies data or changes the values of parameters and observes the effect on the resulting output. These applications fall generally into the category of simulation or computer games. The programs involved are long and complicated. They will not be worked out by the individual teachers—they will be developed by experts. A teacher using them won't have to know anything about the actual programming—he merely has to be confident that they do what they are supposed to. In other words, the teacher using the computer in non-mathematics courses needs to know next to nothing about computing—only how to call up the program and plug in numbers. Contrariwise, the mathematics teacher has to know enough programming to teach it to his students and to help debug their programs.

On the basis of these observations one might suppose that the non-mathematical use of computers, since it is so much less demanding on the teachers, would be much more widespread than the mathematical use. The exact opposite is actually the case. Why? The reasons for this phenomenon are easily identified. They are:

(1) the non-mathematics teacher assumes that any use of the computer involved vast technical knowledge and this inspires him with fear bordering on terror;

(2) he receives no training in computer use in his college training mainly because the education faculty adopts the same attitude toward computing as he does;
(3) the availability of computer oriented materials outside mathematics is still comparatively sparce;
(4) information about existing materials is not readily accessible to teachers;
(5) the importance of computing is nowhere near as great in other areas as it is in mathematics.

To rectify this situation four measures are necessary:

(1) faculties in schools of education must themselves be educated in the use of the computer in their fields;
(2) teachers' institutes featuring the application of computers outside of mathematics must be promoted;
(3) city school systems must hire specialists in non-mathematical uses of computing who can indoctrinate teachers and keep them informed of current developments;
(4) computer oriented student texts and supplements must be produced.

Basically, developments in this area involve a two-stage process—training the education faculty to train the teachers. For this reason developments will be relatively slow in coming unless the National Science Foundation and the U.S. Office of Education decide to initiate a crash program.

The situation is much brighter in the area of mathematics—the reasons being that the effectiveness of computing in mathematics is quite obvious; a large amount of material is available; and the technological aspects of computing hold relatively less terror for the math teacher. The way in which the math teacher has obtained his training may come as something of a surprise. Up to now it has been almost all in-service rather than pre-service training. NSF summer institutes have played a major role here. But in Minnesota, University sponsored summer institutes and academic year seminars have been even more important. Last summer, 60 Minnesota high school teachers participated in a university sponsored summer seminar. Currently 77 Minneapolis math teachers are participating in an institute jointly sponsored by the Minneapolis school system and the University of Minnesota. Better than half the junior and senior high school math teachers in Minneapolis schools have now had computer training. All math students in this school system will have some exposure to the computer.

In outlying areas of the state as might be expected, we see a much bleaker picture. Where the teachers and schools are widely dispersed, teacher training becomes more difficult and the acquisition of hardware presents a formidable obstacle. These problems can only be solved if the state department of education assumes the initiative in forming cooperative regional networks. Such a regional network has been set up in the junior college system. All of Minnesota's junior colleges now have consoles, a development which came to fruition during the current academic year. (As a footnote it is interesting that Minnesota's junior colleges are exhibiting an interesting counter-trend; the chief customers for computing here are the physics courses—the result of a particularly effective training program for junior college physics teachers.)

In spite of this counter trend, the computer in the high school is closely associated with the mathematics program. One reason for this is that elementary mathematical algorithms involve much less extensive knowledge of programming than non-mathematical algorithms and so provide a convenient vehicle for initiating the student in computing. But this is of relatively minor importance. The more fundamental aspect of this interaction is found in the impact of the algorithmic approach in the clarification of mathematical concepts. In college mathematics this impact seems likely to completely revolutionize the teaching of calculus, linear algebra, ordinary differential equations and numerical analysis. And the impact of the algorithmic approach looms no less dramatically on the secondary mathematics scene, so dramatically in fact that there is the temptation to try to use it everywhere, even in those places where it is inappropriate. The great scientist Eugene Wigner has spoken of the "unreasonable effectiveness of mathematics in the physical sciences." In the title of this talk I have paraphrased Wigner's words to "the unreasonable effectiveness of computing in mathematical pedagogy." And indeed this effectiveness is so great as to seem almost mysterious. Now I will try to uncover some of the clues to this mystery which I catalog below.

PRINCIPAL WAYS IN WHICH COMPUTING AFFECTS MATHEMATICAL PEDAGOGY

- Dynamic vs. Static Approach
- Computing Attitude toward Variables
- Development of Problem Solving Technique

- Affinity between Algorithm Construction and Theorem Proving
- Mathematical Topics which are Algorithmic in Nature
- The "Teaching Effect"
- The Stimulation to do Independent Work
- The Excitement of Being Involved in Modern Technology.

Now let's examine and explain the items in this catalog.

Dynamic vs. static approach

In computer oriented mathematics the emphasis is on "what you do to find it" rather than on "what it is." This particularly applies to definitions and existence theorems which can often be put into an algorithmic setting. I believe that the vast majority of students tend to think dynamically and be stimulated by this more active approach.

The computer attitude toward variables

There is a subtle but important difference between the computer attitude toward variables and the traditional mathematical attitude. Traditionally, a variable is a symbol for which values may be substituted from a certain set called the domain of the variable. In computing a variable is a symbol which at any time has a definite value although this value may change from time to time. In the computing approach the student feels that variables stand for something concrete and definite and thus feels more at home with variables and arithmetic expressions.

The profound importance of this subtle distinction was first brought home to me when I was observing a seventh grade class trying out some SMSG text materials on flow charting. The teacher was quite pessimistic about the prospects for success. He said that seventh grade students could understand arithmetic expressions to the extent of being able to substitute values for the variables to evaluate the expression, but that they could not manipulate with the expressions to find other equivalent forms. It turned out that the students had no difficulty with the material but in fact assimilated it at a prodigious rate. I remember particularly vividly the reaction to the simple over-time algorithm in Figure 1 where the variables stand for wage, rate and time. One student commented that the expression $R\times 40+1.5\times R\times(T-40)$ could be simplified to $R\times(40+1.5\times(T-40))$ which would be more efficient as it would require one less arithmetic operation Another student observed that the expression could be still further simplified to $R\times(1.5\times T-20)$. The class agreed with these simplifications and appreciated the reasons for them. The teacher and I were both quite dazzled by the easy familiarity with arithmetic expressions on the part of these seventh graders with no experience in algebra. And this was on the first day of flow charting with numerical algorithms.

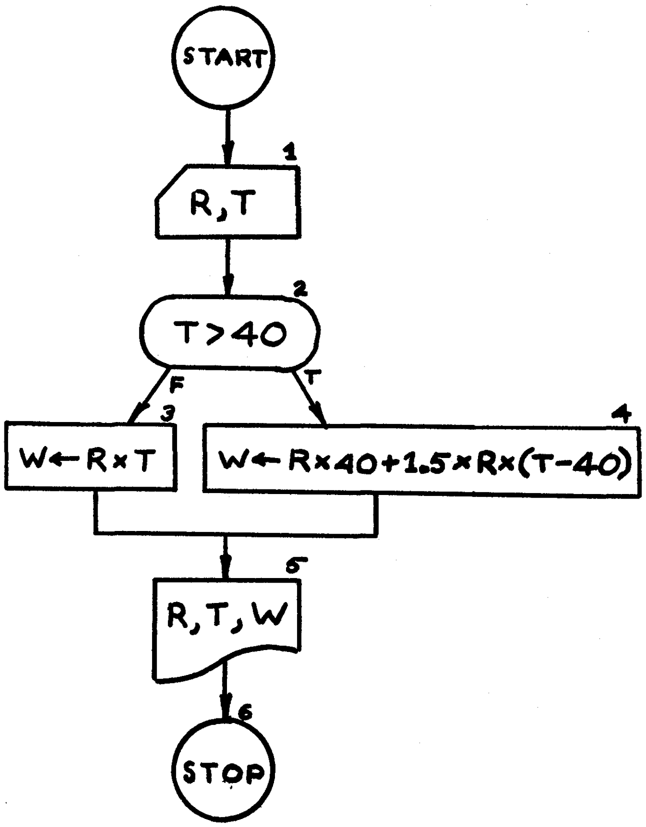


Figure 1—Over-time

One difficulty some students seem to have with mathematics is that arithmetic expressions when they become too complicated have no meaning for them. They see these expressions only as strings of symbols. This applies already to such an expression as the quadratic formula

$$\frac{-B+\sqrt{B^2-4\cdot A\cdot C}}{2\cdot A}$$

In the computing approach they tend to see the expression as being in itself an algorithm for calculating a particular numerical value.

This motivates a remark I'd like to make on computer languages. I have long felt that in more advanced courses applying the computer, such as computer calculus, it is absolutely necessary to use procedural languages like FORTRAN or BASIC. The difficulty with assembler languages is that students get involved in so much detail that they lose sight of the original mathematical ideas. Experiments with teaching computer calculus using assembler languages have borne this out. However, in earlier grade levels where the problem consists of appreciating the meaning of arithmetic expressions the situation is entirely reversed. Here in programming with a prototype assembler language such as SAMOS the student plays the role of a compiler as it were and breaks the expression into its component parts, thus getting a real feeling for the meaning of arithmetic expressions. I hope some experiments will be made to test this pedagogical technique in the seventh or even the sixth grade.

Development of problem solving technique

Most teachers have had the frustrating experience that many students faced with a problem where they can't see the path from beginning to end just can't be made to do anything at all. We just can't seem to get them to pick up a pencil and start writing *something*. Experience with algorithm construction changes all that. Here students learn to find the heart of the algorithm—the fundamental idea—and then to work both ways from the heart. They soon appreciate the futility of trying to start at the beginning and work toward the end. They learn to have patience with their abortive efforts.

Once the student finds a solution, any solution, he stands back and looks for simplifications and improvements. He looks for improvements in the efficiency, the elegance, the clarity of his algorithm. He also looks for additional features he can add to the algorithm, making it do more.

All this is the essence of problem solving and theorem proving in mathematics. The difference is that in the computing setting students take to these activities like a duck to water. The ability to handle hard problems is not confined to the gifted. Many students learn to tackle really difficult algorithms requiring weeks of work, breaking them down into their component parts and putting the pieces back together. As one by-product of teaching computer oriented mathematics we hope to tap this problem solving power and direct it toward solving mathematics problems.

Affinity between algorithm construction and theorem proving

We have already seen in the discussion of problem solving how the mental attitudes necessary for theorem proving are stimulated by the computer viewpoint. But the relation between computing and theorem proving is more than just an analogy. In the computer approach to calculus and linear algebra most of the existence theorems are nothing more than algorithms. "If you can show how to find it you know it exists." This suggests a constructive approach to mathematics as in fact is the case. Although we do not accept all the goals of the intuitionists, we travel a long way down the road with them.

One result of presenting mathematics from a computer approach is that the entire course is reorganized. Although we arrive at essentially the same destination we get there by an entirely different route. The advantage is that the average student will understand more of the theory and the top student will understand it better. There are very definite levels of understanding. It is one thing to follow the steps in a proof. It is another thing to feel it with a sense of inevitability.

These remarks may help to dispel a popular misconception relating to computer oriented math courses. It is widely supposed that in such courses we use calculations as a substitute for theory. In actuality the main function of computing in these courses is to reinforce and promote the understanding of theory.

Mathematical topics which are algorithmic in nature

There are a large number of mathematical topics which are intrinsically of an algorithmic nature. In college mathematics some of these topics are: convergence of sequences, the definite integral, ordinary differential equations, numerical analysis, and practically all of linear algebra. There are also many topics in high school mathematics which lend themselves to algorithmic treatment. Among them are: solutions of systems of equations, the Euclidean algorithm and the fundamental theorem of arithmetic, a wealth of material connected with factoring and primes, area under curves, mathematical induction, sigma notation, square roots by the "divide and average" (Newton's) method. In the traditional ways of teaching these subjects, the algorithmic approach has not been stressed. Most of these subjects have been very poorly understood by students. The algorithmic approach should produce a high level of comprehension. A couple of examples are in order to show why.

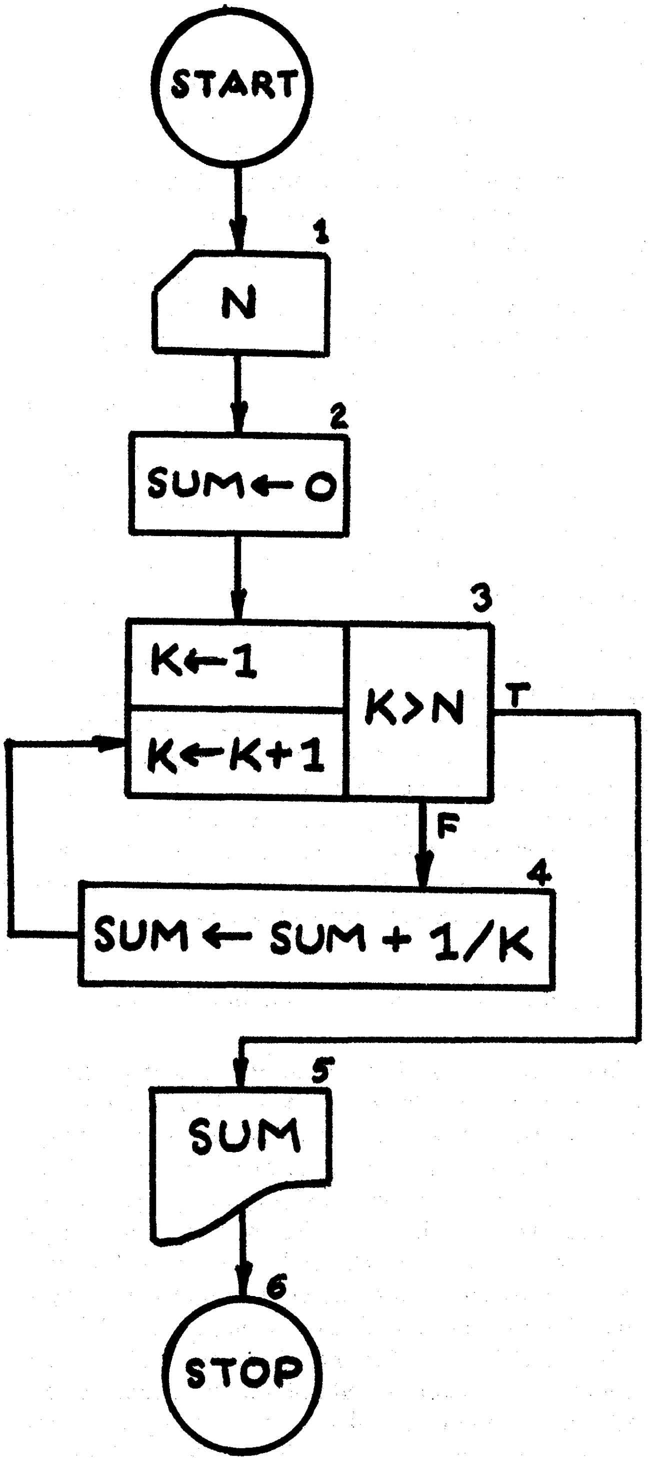


Figure 2—Sigma notation

The sigma notation, as exemplified in

$$\sum_{k=1}^{n} \frac{1}{k}$$

is nothing but a shorthand notation for the algorithm depicted in Figure 2. Students traditionally have a great deal of difficulty with this notation. If you don't believe it, ask your students to find

$$\sum_{k=1}^{100} 1$$

Using the algorithmic approach the student can be asked to trace the flow chart of Figure 2 by hand (with $1/k$ replaced by 1) until he sees what it does.

We see here that our very definition of the sigma notation is as an instruction to perform a calculation rather than as the result of this calculation. This is an example of the dynamic as opposed to the static treatment. Another important point illustrated by the sigma notation algorithm is the concept of a bound or "dummy" variable. The student sees clearly that the final answer has nothing to do with the variable, k. Bound variables as encountered here and as the variable x in the expression

$$\int_a^b f(x)\, dx$$

have always been a stumbling block for students which the algorithmic approach goes a long way toward removing.

For another example of how the algorithmic approach affects the presentation of a mathematical concept we will look at mathematical induction—a disaster area in the math curriculum. Suppose we have a statement with an integer variable n in it. Let's call the statement $P(n)$. Suppose we wish to find the first positive integer n for which the statement is false. This search can be represented by the flow chart of Figure 3. Now further suppose that we know somehow that whenever the statement holds true for one value of n, it will hold true for the next one as well. If this is the case, then once we have passed through the True exit in flow chart box two of Figure 3 we will go through it the next time as well and the next and the next *ad infinitum*. Now is there any way that we could ever leave by the False exit so as to get out of this loop? When this question is asked in class some student will always answer that this could only happen the first time, that is when $n=1$. Thus if we add the hypothesis that the statement is true when $n=1$ we see that we can never branch to a stop; we are in a tight loop; the statement $P(n)$ is true for all positive integer values of n. And this is the principle of mathematical induction.

In the latter example we see a use of a flow chart that will never be converted to a program and run on a computer. We see then that flow charts can have uses quite apart from programming. A recent experiment was performed in Minnesota teaching a mathematics unit to three groups of junior high school students, one with flow charts but no computing, a second with flow charts and computing and a control group taught in the traditional manner. It was found that the flow-charting group showed enormous gains over the control group but that the computing group was not significantly better than the flow-charting group. The scale of this experiment was not large enough to be in any sense definitive. However, the results point out that the main benefit to mathematics lies in the algorithmic approach rather than in having an easy way to get numerical answers. In more advanced courses like calculus and linear algebra the computer is indispensible as we don't feel the necessary conviction until we see the numerical results.

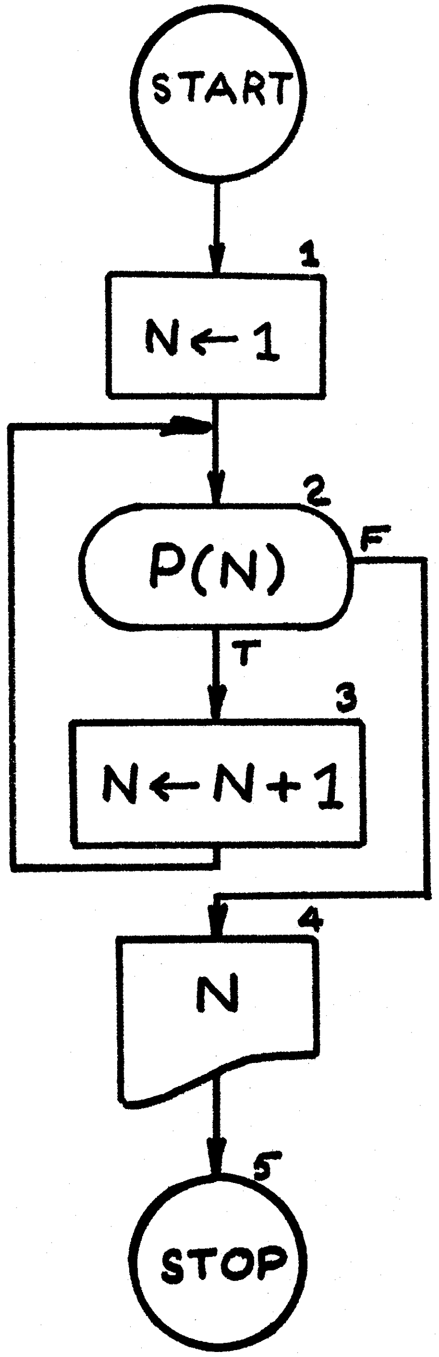


Figure 3—Induction

Computing experience will be of lasting benefit to mathematics students in that they will always have a better understanding of iterative techniques. For example, I have had far better results in teaching Picard's existence theorem in ordinary differential equations (using the successive approximation technique) to students who have been through a computer calculus course.

The "teaching effect"

Every teacher has probably said at some time or another, "You never really learn a subject until you teach it." There is a lot of truth in this. In the computer approach to mathematics the students get the benefit of this sort of experience. For, when he develops an algorithm and writes the program he is essentially teaching the computer to perform the calculation. Actually the student learns much more from this experience than by doing the calculation himself. For his concentration is on the ideas involved in the calculation, where it belongs, and he avoids the tedious arithmetic which is largely counter-productive.

Independent work

Aside from computing I have seen no field of study in which so many students become inspired to invent problems, learn on their own and tackle major projects. This phenomenon has been encapsulized in the phrase "computer bum." Many educators worry lest these students will spend too much time on computing to the neglect of their other studies. However, as I see it, the whole purpose of education is to bring students to the level where they can learn and create independently. We are achieving this desideratum in computing as nowhere else; it should be encouraged and not denigrated. Experts in other disciplines might try to find out why computing is so successful in this respect and try to develop methods of encouraging this sort of resourcefulness in other fields.

The excitement of being involved with modern technology

This item hardly needs any comment except to observe that many educators suppose that it is the only reason for the effectiveness of computing in mathematical pedagogy. Actually it is of relatively minor importance though it is of course a source of stimulation to most (but not all) students and it would be a mistake not to exploit it.

I hope my remarks have shed some light on the present status of the computer in the secondary school—on where it has been and where it is going—and on the reasons for its continuing close association with mathematics in this environment.

Computer-aided design of MOS/LSI circuits

by H. W. VAN BEEK

Texas Instruments, Inc.
Houston, Texas

INTRODUCTION

The functional complexity of MOS/LSI packages has increased exponentially during the past seven years—from a 20-bit shift register with about 130 transistors in 1964 to a one-chip calculator with over 3500 transistors in 1971. This evolution has been realized primarily because of process and engineering advancement in the state of the art. Computer-aided design and layout has played a much smaller part in this development. . . . Why?

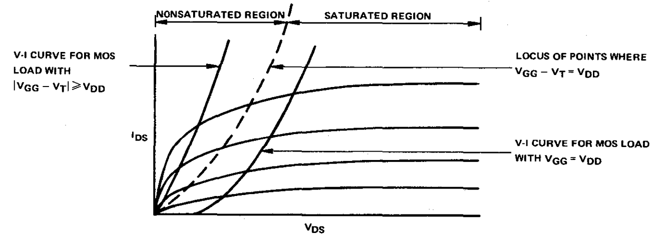
The main reason is that most of the CAD systems developed to date do not provide a viable and economical approach for implementing MOS/LSI designs. Systems currently in use are either too wasteful in silicon "real estate" or require too many error-prone manual operations. In order to achieve greater functional complexities in an MOS/LSI circuit, further development is required in both the simulation and layout of MOS systems. In this paper I will review the current use of CAD tools in both the circuit-analysis and the chip-layout areas. Challenges that must be faced in the future by the programmer in the CAD area will be presented.

COMPONENT LEVEL SIMULATION

The predictability of the MOS transistor in integrated form by means of device equations leads to uncomplicated yet very effective transient-analysis programs.

At Texas Instruments the design engineer uses a parametric circuit analysis program for evaluating cell or gate design based on the standard device equations shown in Figure 1.

It is interesting to note that factors such as speed and power dissipation are directly related to the width and length of the MOS transistor. The program requires as input a wiring node list that describes the interconnection between devices. Process parameters as well as geometrical details of the transistors are data inputs. The designer can also describe particular signal wave-shapes to the program for driving the inputs as well the details of clock signals used for propagating information through his circuit.



NONSATURATED REGION

$$I_D = \frac{\mu\epsilon_o\epsilon_{ox}W}{t_{ox}L}\left[(V_g - V_T)\,V_D - \frac{V_D{}^2}{2}\right]$$

SATURATED REGION

$$\frac{\partial I_D}{\partial V_D} = 0 = \beta(V_g - V_T - V_D)$$

$$\therefore V_D = V_g - V_G$$

$$I_D = \frac{\beta(V_g - V_T)^2}{2}$$

$$gm = \left.\frac{\partial I_D}{\partial V_g}\right|_{V_D} = \beta(V_g - V_T)$$

$$\beta = \frac{\mu\epsilon_o\epsilon_{ox}W}{t_{ox}L}$$

Figure 1—MOS I-V characteristics and basic equations

Transient analysis is performed on the data by generating differential nodal equations, which assume charge balance at each node, and solving these numerically. Prediction and correction techniques are used to select the proper time constants for solving each set of equations.

The program is strictly an analysis tool as opposed to a synthesis medium. Correlation of the computed results compares very favorably with actual results.

The user can specify a tabular listing of absolute and differential node voltages and device currents in addition to a printer plot of voltage versus time.

What is the challenge for software development in this area? I think it is basically that of staying up with the new processes that are coming out of the R & D laboratories and ensuring that the models used for computation remain accurate. The other requirement is to reduce the cost/computer run. Even though the current program executes relatively fast—about 1 minute task time on a 360/65 for a typical problem—it is used so extensively that a 10 percent reduction in run time would represent a sizable monetary savings.

Another aspect is the time required for data preparation by the designer; this, economically speaking, is possibly even more important. With the reduction in cost of simple alphameric CRT display terminals, it becomes feasible to provide each engineering office with such a terminal. The value in this approach, as with other interactive CRT systems, is that during the session at the console you (the programmer) can extract from the engineer his best effort. Input data checking is performed instantly and answers provided when the engineer is most irvolved with solving his problem. This is an extremely important point to keep in mind—software development in this area must be perfectly synchronized with the *modus operandi* of the design engineer. You must anticipate his requirements before he has a change to vocalize them. This implies that you, the programmer, be very conversant in the problem area.

Understanding the problem to be solved for the design engineer is even more important in the area of MOS/LSI chip layout.

COMPUTER AIDS FOR THE LAYOUT OF MOS/LSI CIRCUITS

One of the challenges facing the MOS designer in this area can best be exemplified by discussing a simple but accurate example of a partial system implementation. Logic in the form shown in Figure 2 had been implemented in standard TTL logic; it represented a portion of a larger logic system which was to be converted to MOS.

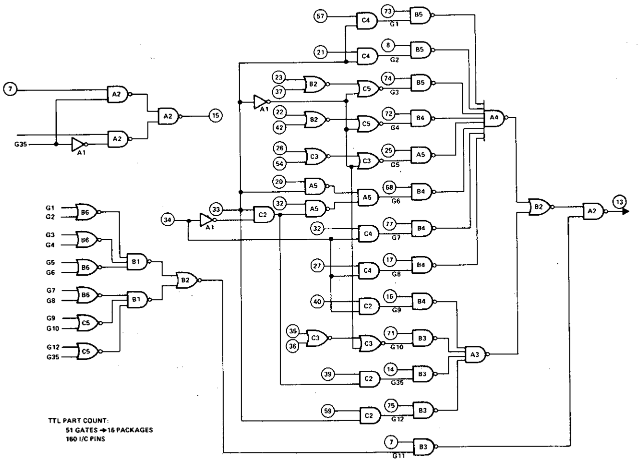


Figure 2—Signal select control logic

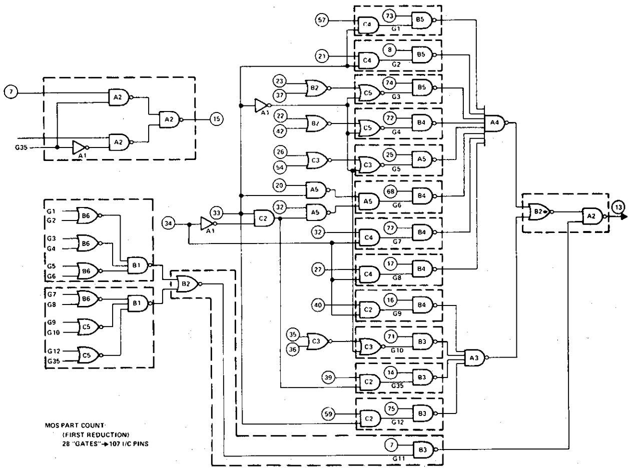


Figure 3—First reduction, signal select control logic

The designer has available to him an automatic P & R (Placement and Routing) program with a substantial number of cells. If he chose this approach he would initially try a one-for-one replacement of the TTL logic diagram with 51 MOS cells. These "library" cells have a standard height of say 12 mils. Assuming an average of three I/O pins per cell and 1.2-mil centerline spacing, we can arrive at an area requirement of 12 mils × 2 pins/cell × 51 cells × 1.2 mils/pin = 12 × 184 mils. This is the area required for the gate (or cells) only, not the intraconnect. For the intraconnect we can assume, for a random logic circuit implemented by means of standard cells, that roughly 20 percent of the area will be occupied by cells and the remaining by intraconnections and bonding pads.

In this particular case the circuit size would be roughly

$$\frac{12 \times 184}{20} \times 100 = 11040$$

square mils or about 100 × 110 mils. The first comment that comes to mind is that the routing algorithm or the designer was not very good at the task. Let's see, however, what the designer can do from this point.

By combining gates that have a fan-out of 1, he can arrive at an arrangement as shown in Figure 3. This approach leads to 28 "standard" cells with a total of 107 I/O pins, resulting in an area requirement of $\approx$7704 square mils or a 31 percent reduction over the first implementation. The designer can continue the "customizing" of cells and maybe achieve a layout equally divided between active and passive area of about 60 × 60 mils.[2]

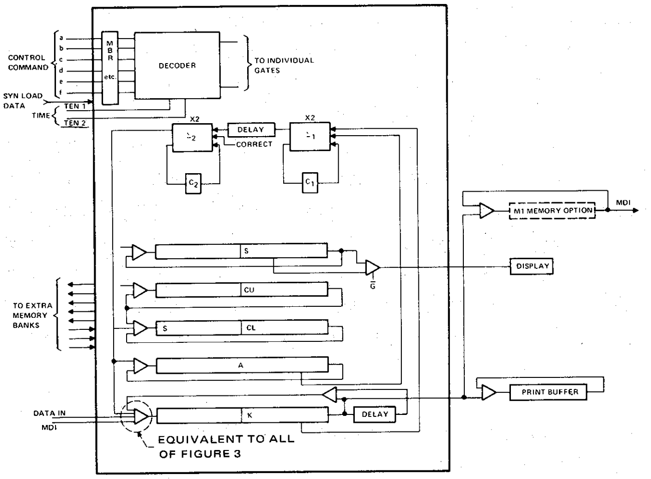


Figure 4—Arithmetic unit

The designer reduced his dependence on the automatic placement and routing program because of the custom design performed, but he now apparently has a small layout. However, this is a false assumption since he only implemented a small percentge of the overall system as shown in Figure 4. Just one small triangle represents all of Figure 3!

Sophisticated system implementation demands a better technique. The designer must be forced to look at the overall logic requirements of the system and break it down into functional blocks. He also must try to make maximum use of the PLA (Programmable Logic Array) or matrix technique since this approach uses the same real estate for active devices as well as for the passive intraconnections to these devices.

The PLA technique is an approach in which system logic equations are broken down into sets of Sum-of-Product terms. Implementation follows by generating Product terms in an AND matrix, and summing these in an OR matrix to produce the desired output (see Figure 5). Memory elements can be added outside the matrices to satisfy time delays and/or storage requirements. Almost all random logic circuits generated at TI contain one or more PLA's for control functions.

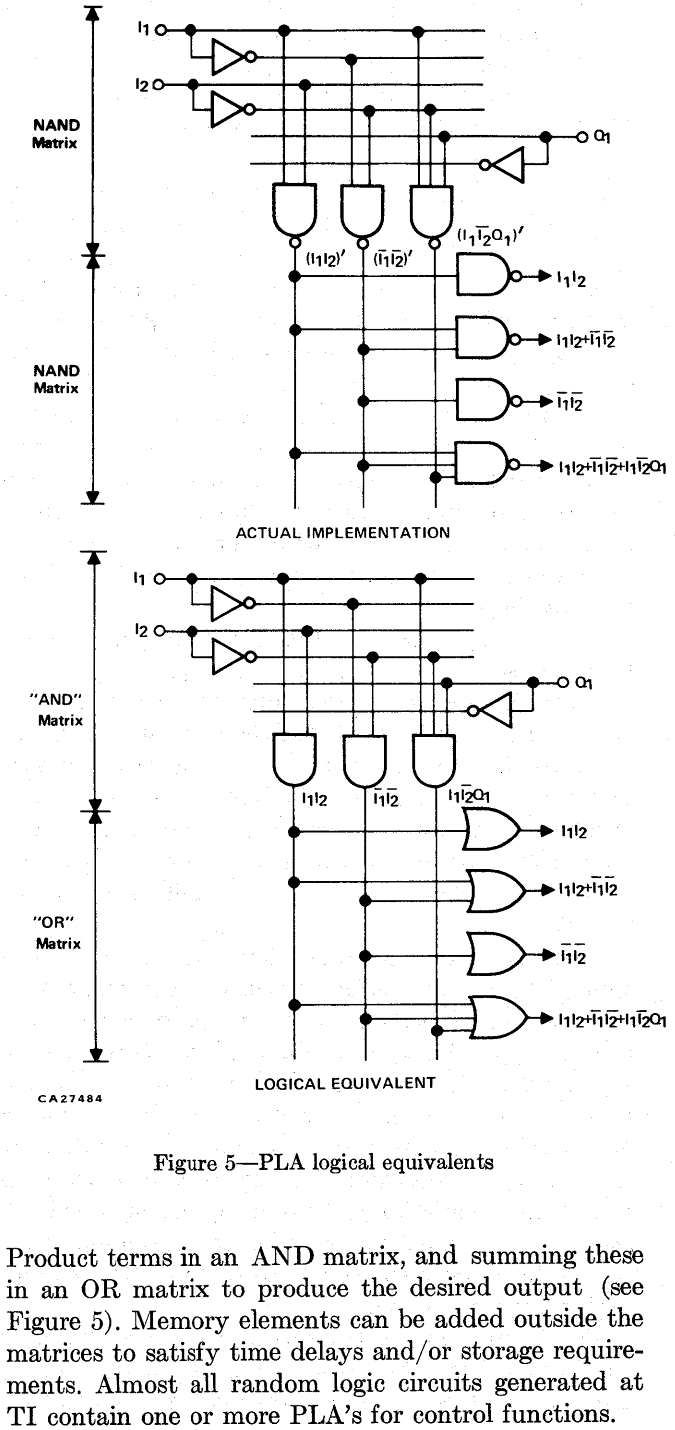


Figure 5—PLA logical equivalents

What is the future then for CAD if we suddenly discard the standard gate or cell approach in favor of a technique that doesn't lend itself to the nicely organized world of fixed-height cell layouts? Is automated placement of cells by means of a program obsolete? I believe that current versions are obsolete for most implementation requirements because of the poor area utilization. The human skill at conceiving an essentially infinite number of possibilities for placing arbitrary size polygons in a minimum area is one that is too costly to implement automatically. For an equivalent logic function the skilled draftsman can always generate a smaller layout than can an automatic placement and routing program.

Let's take a look, however, at the other side of the coin. Future demands on the functional complexity of packaged MOS circuits will increase by leaps and bounds if the past few years are any indication. These pressures will lead to circuit complexities that cannot be handled by only one or two designers, but will require teams of engineers. Even so, the chance for error during the design and layout process will be so great as to make the end product too expensive for the market place. One missed connection or contact on a circuit can cost thousands of dollars in lost time and effort to correct and recycle. An effective means of minimizing errors during the design and layout of a circuit is therefore mandatory.

It is this aspect of CAD that is currently undergoing the greatest evolution. Interactive-graphic systems are now becoming available that allow the designer to lay out his circuit under control of the computer. Data are input to the system by means of a digitizer, graphic tablet, joystick or light-pen. This is an effective manner of reducing the errors that occur in cases where layouts are implemented entirely by hand, because a complex circuit can contain over a quarter million coordinate points!

An interactive-graphic system in use at Texas Instruments lets the user perform his cell placement and the intraconnect between cells, by means of a light-pen-controlled CRT. Major program functions such as signal routing and placing cells are called up by means of a function keyboard. The program has automatic layout rule verification and demands an override command from the operator in order to proceed with the layout. Automatic conductor-spacing routines facilitate the layout of large groups of bused conductors. Only cell outlines are displayed to keep the data handling problem to a minimum.

This system allows the operator a great degree of freedom in doing the work he is most qualified to do, namely the placement of arbitrary-sized cells and their intraconnection. The system meanwhile performs a clerical function—that of encoding the light-pen and function-keyboard commands. The encoded data is in a form suitable for driving digital plotters, and mask-generation equipment.

DATA VERIFICATION

The question that now arises is: "Can we trust that no errors are made in the layout?" The answer is NO, because the user can usually override the system error diagnostics. How do we check his work? A special layout verification program was written to check the final layout just prior to the mask-generation step. It will check all circuit elements against each other for layout rule violations. This task was previously performed by a visual inspection of a digital composite plot of the circuit but the amount of data was becoming so large that errors could and did slip by.

The Critical Clearance program checks both the dimensional consistency of the data input as well as the spacing between figures. This check is performed on data of a particular mask level only. An extension of this program performs a level-to-level check and verifies proper layout rule overlap requirements. Continuity checks are also made to detect open lines or incomplete transistors that might exist in the data deck. A trace-back to the circuit schematic and the original logic description completes the feedback cycle.

FUTURE DIRECTIONS IN MOS/LSI IMPLEMENTATION

Respectable advancements have been achieved in the area of CAD and layout of MOS circuits during the past 5 years, but even so they will not be able to effectively help in our future designs. There still exists a large gap between the abstract aspect of a logic diagram and the concrete point of a circuit layout that is currently filled in with manual and error-prone procedures. To be more specific we must devise a system that accepts logic equations in an existing or new format and provides us with an optimum topological layout. This requires that a component or circuit data base be established with which the designer can interact.

Note that I mentioned a component or circuit data base as opposed to a circuit or logic system data base. I propose that effort be directed toward developing such a data base and that placement and routing algorithms be developed for it. The availability of such a system would allow the MOS designer to bypass the generation of cells for his circuit layout and consequently permit him to generate a circuit directly from his verified logic.

Additional programs that would complement such a system will accomplish the following:

1. Full-circuit parametric evaluation.
2. Minor adjustments to circuit layouts by means of an interactive graphic terminal to accommodate process changes, and
3. Generation of documentation pertaining to the circuit such as bonding diagrams, process flow, parametric test criteria, packaging information, etc.

The above steps currently occupy a sizable portion of the cycle time for completion of a prototype design. The variety of circuits that will be designed during the next few years will bury the component manufacturer unless he takes decisive steps in the aforementioned direction.

The role of simulation in LSI design

by J. J. TEETS

IBM Systems Development Division
Endicott, New York

INTRODUCTION

Forecasts for Large Scale Integration (LSI) indicate that logic gate densities of 1000 gates per chip[1] could possibly be obtained during the next decade. Supposedly, device and circuit innovations could be used to achieve even higher densities. There are, of course, a number of reasons why the industry is not yet manufacturing major LSI hardware on a large scale. One of these reasons is the lack of superior software support for computer-aided design, particularly simulation aids. For LSI, a major increase in logic design verification simulation will be required at the chip level to ensure correctness of design and to minimize the need for chip redesign.

The purpose of this paper is to review what has happened in the last five years, particularly in design verification, and to project what must be accomplished in the next five to ten years in the area of simulation in order to implement the promise of LSI.

REVIEW OF THE LAST FIVE YEARS

This section briefly describes high-level (architectural) and low-level (circuit component) simulators, and then closely examines recent logic level simulators. Logic simulation employs much more detail than high-level simulators and much less detail than circuit simulators. The purpose here is not to demonstrate that these simulators are inadequate for LSI technologies, but to show that each still has a place. The intent is to display a need for multi-mode simulation at the logic level in one user interface.

High-level and low-level simulation

Predictably, there has been and will be high-level simulation studies on various approaches to implementing LSI. Some of the major aspects and tradeoffs evaluated include: (1) replacing third generation systems with LSI, (2) enhancing architectural approaches with LSI scratch pads and main store, (3) microprogramming techniques (both read-only and writable control stores), and (4) considering hardware-software tradeoffs.

An example of an available capability for such high-level studies is the General Purpose Systems Simulator (GPSS).[4] GPSS provides for modelling systems and can collect various statistical measurement quantities such as throughput, resource utilization, queue lengths, busy times, system bottlenecks, etc. The collection of such data is very valuable in evaluating high-level architectures, both hardware and software, in all phases of design.

On the other end of the simulation spectrum is the very detailed low-level (i.e., circuit analysis) simulator. This analysis is at the component junction/resistor level; here the differential equations are solved by numerical methods.

Design verification at the logic level

The requirements of pre-LSI technologies such as RTL, DTL, TTL and recently ECL have led to the development and enhancement of computer-aided design support for simulation, wiring, testing, and packaging of designs. The concept of a central design file, used by all these applications and massaged to satisfaction, is well-known.[3] In this section, the simulation techniques used for technologies leading to LSI are reviewed in some detail to provide a basis for discussing later improvements and extensions required for today and tomorrow.

The techniques used by early logic design verification simulators were of relatively narrow purpose, i.e., they attacked a small set of design problems by using techniques specifically designed for these problems.

There were basically three early types or modes of

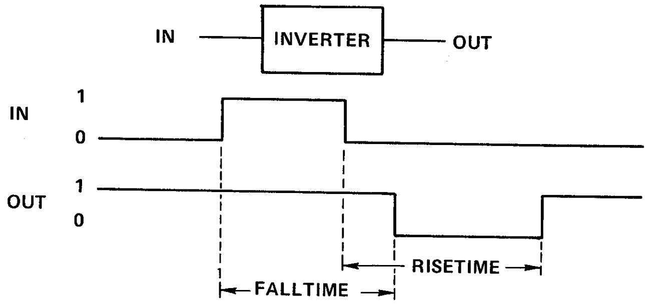


Figure A—2VND simulation

logic simulation that became popular and were implemented (2VND, 2VUD, and 3VZD). All were similar (table driven, handling both sequential and combinational logic) but each was valuable to the designer in a different way.

1. 2-Valued, Nominal Delay Simulation (2VND)[3,9]
   This mode of simulation assumed that the state at any point in the logic at any given time was either "0" or "1". The nominal delay was a gross approximation of the circuit delay (including loading and line delay where appropriate) in the form of a "rise" or "fall" delay, which corresponded to the amount of time taken for the circuit to respond to an input change and change the output state from 0 to 1, or 1 to 0, respectively (see Figure A). When primary inputs to the logic are stimulated, the simulator performs the Boolean function of the blocks driven by the primary inputs. The basic functions performed are shown in Figure D (ignore the "X" and "U" values). For each block whose value is to change, the proper delay is selected and the new block value is propagated after that delay. Propagation continues through the logic until a steady state is reached. When a block's value does not change as a result of an input change, propagation does not occur. This is known as "significant event" simulation and is common to all three simulation techniques being discussed. This mode of simulation, although somewhat gross in timing analysis detail, was very useful in gaining some engineering confidence in the behavior of the early design. This mode was more economical than the "three-valued zero delay simulation (3VZD)."
   One area where this mode fell short, even to the point of misleading the user, can be demonstrated by the following example: the 2VND technique (as used in early simulators) predicted that the response of a TTL inverter whose rise and fall delays were 15 nanoseconds (ns) to an input pulse of width 1 ns, is an output pulse 1 ns wide and 15ns later than the input (Figure B). In fact, however, a TTL gate requires a minimum input width of about 12ns to achieve any output response and would, therefore, completely ignore an input pulse of 1 ns or even 10ns wide (Figure C). Moreover, the technique is not pessimistic as a simulator should be. It allows the user to think that he can count on a narrow pulse to get through a logic gate that is sure, in fact, to ignore it. The point here is that simulators should not tell lies. Pessimism may produce conclusions from simulators which may never exist in real hardware and hence cloud the issue; however, the designers must be alerted to potential problem areas, and then decide for themselves when a real problem exists.

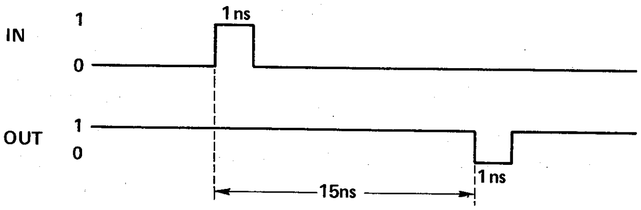


Figure B—2VND prediction of TTL inverter response

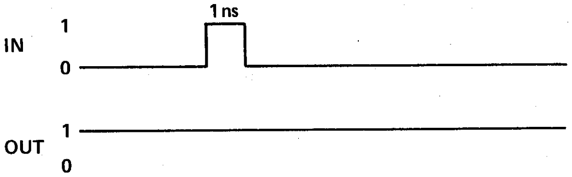


Figure C—Actual response of TTL inverter

2. 2-Valued, Unit Delay Simulation (2VUD)[5]
   This mode is very similar to the 2VND mode, except that the delay of each circuit is the same (one unit), independent of the type of circuit and the direction of change. The contention of this mode is that if one unit of delay is assigned to each logic block (with some exceptions which are assigned a delay of zero), the results of simulation will be sufficiently accurate to serve

a very useful purpose. Furthermore, if a convenient facility is available to add or reduce delays by units, the simulator will be able to closely represent the operation of the actual machine. The applicability of this approximation is dependent on the manner in which the machine was designed. Due to the variation of delays in logic circuits from manufacturing, the designer must allow sufficient tolerance so that all of the manufactured machines can operate identically to the prototype within certain limits of delay variations. Thus, one can consider the unit delay machine merely as one of those variations.

The most important advantage for this simplified timing simulation mode is its speed. It is faster than the other modes and does not require knowledge of nominal delays nor does it require storage for those delays in the simulator. It is subject to shortcomings similar to 2VND in that both modes fail to detect most hazards (subtle timing problems).

3. 3-Valued, Zero Delay Simulation (3VZD)[2, 9]

This mode, the most pessimistic of the three presented, was implemented primarily to detect hazards and races,* as well as to investigate Boolean behavior. All possible timing problems of a design are identified to the designer. This was realized by the introduction of a third value, "X". Any time an input (beginning of course with the primary inputs) changes state, it is forced to pass through X, signifying that the signal is "in transition" or "unknown". When this technique is used, first every primary input that is changing state goes to X, the driven blocks are simulated, and propagation of all blocks to a steady state occurs. The rules for simulating X in a Boolean function are a subset of Figure D, ignoring the "U" for now. After a steady state is reached, the changing inputs go to their final values and again propagation takes place to a steady state (if possible). If a timing problem is a possibility, the value remains X on a block. This indicates a potential hazard involving that block. The possible steady state effects of propagating the hazard are also shown by X values. A facility exists to allow the designer to eliminate some of the simulator's pessimism where races detected by the simulator are known to be noncritical. When used properly, this mode will allow the designer to be confident that all potential timing problems have been identified. However, eliminating pessimism in one part of a design often has unexpected effects on other parts.

Inputs		Outputs			
A	B	AND	OR	EXCLUSIVE-OR	INVERT $\bar{A}$
U	U	U	U	U	U
U	X	U	U	U	U
U	0	0	U	U	U
U	1	U	1	U	U
X	U	U	U	U	X
X	X	X	X	X	X
X	0	0	X	X	X
X	1	X	1	X	X
0	U	0	U	U	1
0	X	0	X	X	1
0	0	0	0	0	1
0	1	0	1	1	1
1	U	U	1	U	0
1	X	X	1	X	0
1	0	0	1	1	0
1	1	1	1	0	0

Figure D—Example of simulator rules
(for all modes, 4-valued)

Because of the two-pass operation described above, this mode of simulation is a little more expensive to use because of the additional computer running time; however, this must be evaluated as a tradeoff against extensive hazard detection capability not available with the other modes.

4. Extensions

Combinations and other variations of the previous three techniques have been used;[6] one of the more useful perhaps was the introduction of a fourth value, "U", which signifies "uninitialized". When a U appears on the input of a block

* If it is possible for the output signals to behave in a manner different than predicted, the circuit is said to contain a *hazard.* When two or more signals are changing together a *race* is said to exist (Eichelberger).

being simulated, it is propagated according to the rules of the simulator (Figure D). At any given steady state condition, the appearance of U's in the logic indicates the absence of initialization data, and therefore, identifies the importance of initial values. This can be very useful when evaluating system reset operations or the effect of incompletely specified input patterns. Two forms of the output from these simulators include: (1) timing charts, which show the reaction (in simulated time) of inputs and outputs of selected blocks, and (2) "snapshots", which display the value of all points in the model at selected times.

Other improvements include extension of 2VND and 2VUD to include both the X and U concepts. Thus, the nominal delay and unit delay modes can be made more useful in the area of hazard detection.

5. The Unfortunate User

Irrespective of the merits of these simulators individually, they had the collective disadvantage of not being mutually compatible. To simulate for more than one purpose (using more than one mode) required considerable, if not total duplication of effort. The problem was compounded because each program had a unique user interface. Thus, the user had to learn several incompatible systems before he could simulate at any desired point in his design cycle. The cost of this additional training and duplication of effort is no longer tolerable, particularly for LSI. It is now essential for different simulation efforts to have maximum commonality in all respects, particularly the user interface. This resulted in the requirement for multi-mode simulation and leads us into present simulator technology.

TODAY AND TOMORROW

Current simulation techniques in the design process

Whatever the size of a design project, there is a general design procedure to be followed. Several stages in that procedure involve simulation and these and other stages may be iterated to feed back information to improve the design. For purposes of discussion, these key stages in design are:

1. Establish system objectives and specifications.
2. Model the system and evaluate it by high-level simulation.
3. Partition the system into units (subsystems) and establish unit specifications.
4. Model the units and compare to specification by simulation.
5. Design units at the implementation level, modelling them for design verification simulation.
6. Compare models from different stages by simulation to ensure the detailed design meets unit and system specifications.
7. Build the system.
8. Check the system against simulation results.

At stages 2, 4, and 6, simulation is employed; however, it is obvious that the level of system being modelled is different at each stage. Moreover, stage 6 simulates several levels together. Hence, multi-level simulation is required for a fully integrated simulation capability.

Another major simulation problem that can and must be solved is that of getting very large models into relatively small computer storages. Previously, users had no option (other than not simulating) except to eliminate bits of their model until it was small enough to fit. This usually caused distress by ending up with functionally awkward pieces and unchecked interfaces between pieces. The solution for this is to partition a model, from the start, into a set of coherent regions (units), each having a well-defined interface with its neighbors (Figure E). These regions may then be simulated together as a system with the simulator automatically handling inter-region transactions by connecting together signals with identical names. An incomplete (or not yet designed) region can be simulated by using manually generated data or higher level models

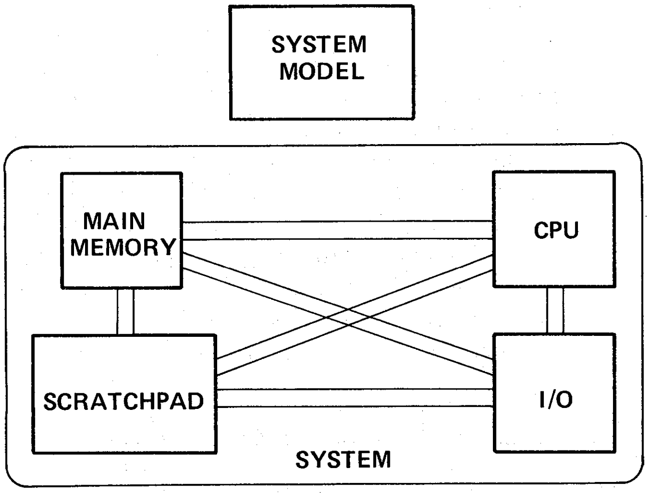


Figure E—System partitioned into regions in place of missing parts which influence the regions modelled at the logic level.

This multi-region concept readily lends itself to techniques for solving the storage problem previously mentioned. One solution is to use a roll-in, roll-out technique to make storage available for active regions. Another solution is to compress the storage requirement (and execution time) of those regions that are being used only to generate interface activity for the benefit of simulating other regions in detail. This compression takes the form of higher level models for the other regions so that the functions can be reproduced at the interface. With this technique, a "multi-level simulation" capability is defined. Merged with a multi-mode capability for those regions modelled at the logic level, an overall capability for system simulation can be provided where the detailed simulation takes place in the area of user interest.

These concepts lead toward the idea of "top-down" design using simulation. In stage 4 of the design, high-level models for the different units may be used to compare specifications of the units against system level specifications by simulation. As stage 5's logic level model becomes available for a given unit, the simulation at stage 6 may take advantage of the higher level models of other units not yet designed in detail. The stimulus (i.e., the input patterns) for the detailed model can be provided by the high-level models surrounding it and can operate concurrently with the high-level models. The high-level model for the region being simulated in detail may be simulated at the same time, and the outputs of both can be fed into a comparator to ensure that the implementation design meets unit and system specifications. This use of the "high-level environment" provides the user a pseudo system simulation at all times, regardless of which unit is being investigated and designed in detail. It also eliminates much of the problem of providing adequate stimulus to the low-level design, which is otherwise a very tedious task.

Thus, the initial and unavoidable need to partition a model can be turned to an advantage. It leads to orderly design, region by region, with thorough checking and evaluation of each unit. In addition, very important advantages, particularly in the LSI design environments, can be addressed in the area of design verification; these include:

1. verify the correctness of design (including comparing high-level to low-level models and comparing different versions of a unit).
2. detect hazards and races which would not be detected by building a hardware model.
3. speed up the design process by using top-down design techniques (which also provide good communication between unit design groups).
4. reduce the overall cost of the design process.

All of the above, particularly the cost factor, are imperative in the LSI design environment.

It has been stated that for LSI, a major increase in logic simulation at the chip level will be required to insure correctness of design and to minimize the need for chip redesign. The engineering change problem is particularly perplexing and still lacks good proposals for its solution. If a chip change is required sufficiently early in an LSI machine project, redesign costs will be incurred although slippage may not occur. During machine test, however, where in the past the practice was to install an immediate fix if possible, slippage as well as redesign costs will accrue; a repeated string of such change cycles is obviously intolerable. Thus, LSI ideally requires error-free design which necessitates simulation aids and capability at a level never required or achieved in the past. Such ultimate goals as self-diagnosing and repairing machines in LSI cannot possibly be achieved without significant advances in design verification and fault simulation capability.

Thus, because of the cost factors and the related problems of system prototype "bring up" (engineering change activity), it appears that the LSI technology is insisting upon much more extensive simulation and paper design before manufacture, and at the same time, a reduction in the overall machine development cycle and cost.

Another salient point that must be made is that it is imperative for fault simulation to be integrated more into the design verification process. If a machine design does not lend itself to system fault diagnosis and component fault detection, then many problems arise. For this reason, the requirements of testing the machine must be fed back into the design process as early as possible. One approach is to use the same stimulus used by the design verification simulators as test patterns for fault simulation. If the patterns detect only a small percentage of faults, then an incomplete job of design verification or an untestable design may be indicated. Quite possibly the answers to testability and diagnosis reside in performing fault simulation as soon as possible after a logic level design is firm, or even during the comparison of alternative design approaches.

The change in engineering design philosophy is largely due to the inadequacies of past diagnostic capability. Tests were derived in the past, in some cases, after the machine had been built. They were, as a consequence, not fully effective in their coverage. The obvious solution is to constrain the initial system and engineering designs by the diagnosis requirements.

An open-ended list of future goals can now be stated for implementation of LSI in the next decade; they include:

1. techniques and software for verifying that system and architectural specifications have in fact been implemented in a machine.
2. highly interactive, terminal oriented, and integrated computer aids for design to reduce user training and turnaround, resulting in usability.
3. advanced and enhanced simulation techniques for verification of designs; particularly multilevel simulation, logic-level simulation, and more effective techniques for timing analysis, at reduced cost.
4. advanced testing capability.
5. a "paper design" capability using software which provides *very* high engineering confidence and paves the way for the reduction or elimination of engineering change activity, at reduced cost and on a shorter design cycle.
6. Automatic input generation (particularly test pattern generation) and automatic output analysis facilities to relieve the burden of the designer.

REFERENCES

1 M E CONWAY L M SPANDORFER
A computer system designer's view of large-scale integration
Proc AFIPS Vol 33 pp 835-845 1968

2 E B EICHELBERGER
Hazard detection in combinational and sequential switching circuits
IBM Journal of Research and Development Vol 9 No 2 March 1965

3 P W CASE H H GRAFF L E GRIFFITH
A R LECLERCQ W B MURLEY T M SPENCE
Solid logic design automation
IBM Journal of Research and Development Vol 8 No 2 April 1964

4 *GPSS V User's Manual*
Prog No 5734-XS2 (OS)

5 L H TUNG
A unit delay logic timing simulator
IBM Systems Development Division Technical Document TD 01.509

6 M J KELLY
Variable mesh simulator
Digest of the Second Conference on Application of Simulation pp 294-296 December 2-4 1968

7 M A BREUER
General survey of design automation of digital computers
Proceedings IEEE Vol 54 No 12 December 1966

8 K A DUKE H D SCHNURMANN T I WILSON
System validation by three-level modelling synthesis
IBM Journal of Research and Development Vol 15 No 2 March 1971

9 S A SZYGENDA D M ROUSE E W THOMPSON
A model and implementation of a universal time delay simulator for large digital nets
Proc AFIPS Vol 36 pp 207-216 1970

10 Proceedings of Hawaii Conference of System Science and Cybernetics 1969-1970

Implementation of a transient macro-model in large logic systems

by N. B. RABBAT and W. D. RYAN

Queen's University
Belfast, U. K.

INTRODUCTION

In large circuits, we are confronted with two problems. First, the modelling of integrated circuit modules using the normal methods of considering each element such as the general circuit analysis program—precise though it may be—can be excessively lengthy and tedious because of the computation time involved and the large storage required.[1,2] On the other hand, since most very large circuits are likely to be active, and therefore non-linear, and since the processes of non-linear analysis are more time-consuming than the corresponding processes of linear analysis, the time required for the inversion of these large matrices would become an even smaller proportion of the total. Though suitable for discrete circuit analysis, this method applied to integrated circuit configurations[3] such as modified TTL and DTL gates leads to complex forms of modelling rendering it inappropriate for use in large systems consisting of hundreds of logic gates. Also, since the models of the elements constituting integrated circuits are more complex in nature than their discrete counterparts, a model leading to accurate predictions of performance will tend to give an excessively lengthy analysis. The choice of models becomes even more critical as the complexity of integrated circuits becomes larger. Even with the recent advances in sparse matrix, adjoint and implicit integration techniques, these methods are not powerful enough to solve the problems associated with large complex logic systems.

Secondly, computer-aided analysis in digital systems commonly involves logic simulation[6,7] and an assessment of propagation delay which is made through the time diagrams for each module available from the manufacturers. However, propagation delay t_{pd} obtained by algebraic methods,[6,7] is imprecise since some important factors governing t_{pd} are not considered. Further, in a logic system the outputs of the modules are delayed and may not be simultaneous even though the inputs are. At high speeds, interconnections introduce significant delays and act as transmission lines; these effects, which cause functional errors and hazards in sequential circuits particularly, are included in the process of simulation.

MACRO-MODELLING

As medium and large scale integration (MSI and LSI) become commercially feasible, the transient simulation and accurate evaluation of t_{pd} in large systems become increasingly important. The need for the accurate prediction of the delays, where complex factors are generally ignored, results from the inability to specify exactly when a given signal will actually make a transition from one state to the other. These delays may also introduce functional errors in the system. Apart from the elemental components and considerations of fan-out, the important factors governing t_{pd} are the rise and fall times of the input function, multiple inputs, short input pulse width, multiple paths, feedback loops and the characteristics of the interconnecting paths. However, to obtain a precise value for t_{pd} and to predict the transient waveform at a given point in a system with multiple interconnecting and feedback paths, a transient Macro-Model is necessary. If the transient behavior of each module under the operating conditions is not precisely known, the actual output logic function obtained may be unrealistic. A novel approach—Macro-Modelling[3,4]—which takes into account all the considerations outlined above and enables a good prediction of waveforms and propagation delays in large integrated digital systems, is based on an analytical formulation of the input-output relationships in transient of the logic module, using the piecewise linear transient device model introduced by Narud et al.[8] Examples of such piecewise linear analytical expressions can be found in References (3) and (8) for a large variety of bipolar and MOST gates.

Koehler[5] alludes to macro-modelling while investigating the effect on propagation delays using quantized tables and taking into consideration the effect of fan-out for the two values of propagation delay for rising and falling edges. He points out the possibility of using empirically-derived expressions for specifying the input-output relationship. Narud, Dertouzous and Jessel[2] have made a good attempt to model an ECL gate by using non-linear input and output current generators and capacitances but their approach has some limitations: the number of gates in the system is restricted to 12 and it is not suitable to analyze systems with saturating bipolar gates or MOST gates as well as interfacing with different kinds of gates. The effect of feedback paths and interconnections causing multiple reflections could not be readily simulated by their method. Macro-modelling in Large Scale Integration (LSI) of digital systems makes use of the periodic nature of a large number of subnetworks. Integrated digital circuits consist normally of a large number of internal nodes—hence variables—whose values need not be known for the evaluation of the overall integrated circuit provided, of course, that their effect is taken into account. Obviously, there are two main aspects of the analysis:

(a) the modelling of different kinds of modules which the system may comprise and
(b) the actual simulation of the whole system.

MACRO-MODELLING IN SIMULATED TIME

The input function is first determined by establishing the dominant input in relation to the other inputs.[4] The dominant input, in the case of saturating multi-input gates, is defined as being the latest arriving and first decaying, or vice versa, dependent on the kind of logic operation performed. For non-saturating gates, a separate integral is solved for each input and the appropriate value is used as the argument in the output function. The dominant input is then subdivided into a given number of fixed time intervals Δt and the output response is computed by successive iteration for each time interval using the analytical input-output relationship for the particular module. The quantized values for the input and output functions at the beginning and end of each interval provide the initial values and slopes (which together form the initial conditions) and must be included in the computation. The simulation is performed by successive iteration of the set of mathematical expressions describing the type of module under consideration. The type of logic module is simulated accurately by an analytical transient input-output relationship—an expression which is analytically derived[2,3,4] and stored for reference. It includes the transfer function of the module as well as a function defining the propagation time. These transient expressions[3,8] are not difficult to obtain using the piecewise linear device model. Each such iteration determines the value of the output for each corresponding equation as a function of the values of the composite inputs. This is illustrated by considering points on the input and output curves (Figure 1) corresponding to time t_i; the time at the beginning of the previous interval is t_{i-1} and $t_i = t_{i-1} + \Delta t$. The value of the output function V_i at t_i is calculated from the corresponding value of the input function I_i and the entire response is obtained by iterated computation. It is important that the current input and output function values (I_i and V_i) as well as the values corresponding to the previous interval (I_{i-1} and V_{i-1}) are made available at the beginning of each interval, since these determine the initial conditions—the initial values and the slopes for each Δt.

Given V_{T1} and V_{T2}, the threshold points associated with the rising and falling input edges, the output response of logic gates may be looked upon as consisting of three main regions:

(i) the region before V_{T1},
(ii) the region between V_{T1} and V_{T2} and
(iii) the region after V_{T2}.

Each main region may consist of one or more subregions. Logic gates normally require one or more subregional expressions[3] to specify their transient response. While each subregion may be considered as a continuous function, the interface point where a subregion begins is defined as a discontinuity. Because of

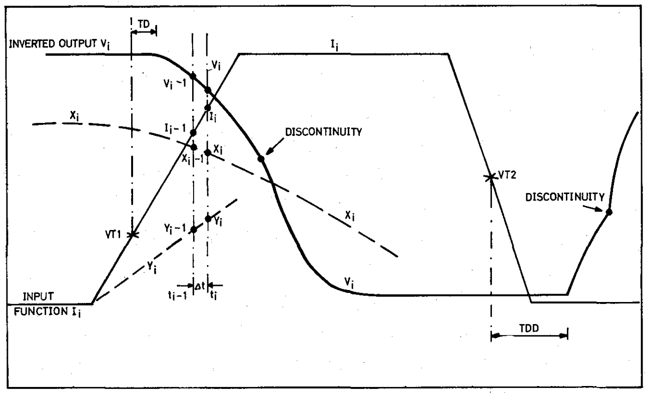


Figure 1—Macro-modelling and waveshaping of continuous expressions for the rise and fall times with discontinuities

the discontinuities associated with the output of these gates, they may be referred to as multiregional gates. Examples of multiregional responses are the modified bipolar and MOST gates.[3] The input-output conditions and transient relationships for each region are different and are considered separately in the simulation. This is consistent with the principle of piecewise linear analysis.[8]

For the output region preceding V_{T1}, an important consideration is the comparison of the slope of the input function at each time interval Δt with the slope of a curve Y_i, which is governed by the internal time constant associated with the gate input.

In general the region between V_{T1} and V_{T2} is governed by TD, a delay time elapsing after V_{T1}, and by the relevant transient relationship of the gate. TD depends on the relationship in time between the rate of rise of the input I_i and the inherent time constant Y_i associated with the gate input. For similar gates, the variation in TD is of little concern, but in a system employing interfacing with different gates, the changes in the value of TD and in the position of V_{T1} must be taken into consideration. Given that I_i is much faster than the internal time constant Y_i, TD is determined by Y_i; whereas, if I_i is slower than Y_i, TD is governed by the input. The required input relationship at interface points is established and the required TD computed. This region is also governed by an intrinsic curve X_i derived from the internal switching characteristics of the gate.[2,8,9] A comparison of the current values (X_i and V_i) at each Δt detects the presence of a discontinuity in the response. The value and slope at the point where a discontinuity occurs are stored as new initial conditions for the next continuous subregion.

The region after V_{T2} which may also be multiregional is governed by TDD, the storage time elapsing after V_{T2}, and the appropriate input-output transient relationship. TDD is an important parameter and governs the output response of t_{pd}. An appraisal of TDD is therefore included in the macro-model.

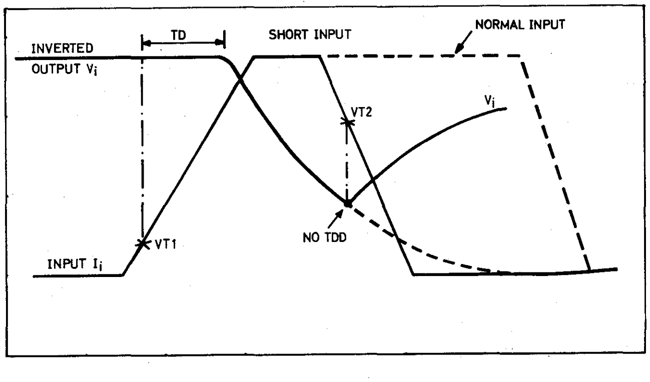


Figure 2—Effect of a short pulse width on the rising response

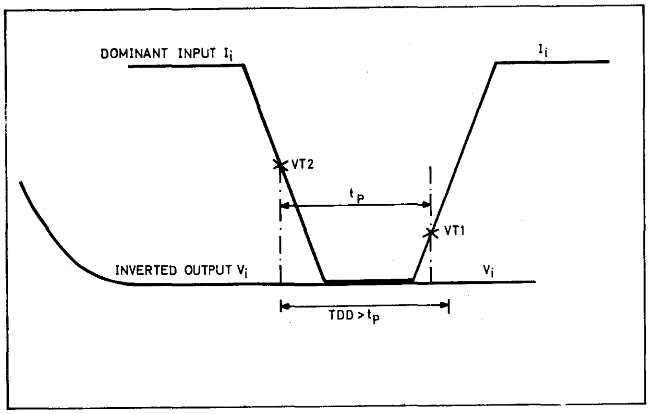


Figure 3—Effect of a short pulse width on the storage time

Assuming that the input waveform consists of reasonably wide pulses, the corresponding output waveform V_i is predicted in a normal manner by considering the parameters TD and TDD for the logic module. If, however, the duration of the input pulse I_i is short and comparable with TD and TDD, then prediction of V_i must be based on the consideration whether the input levels turn the output stage ON or drive it into saturation. It may be seen that this consideration determines whether the output is a pulse waveform or consists of partial pulses or humps. This assessment is made with accuracy by the simulation method, which computes the initial conditions (V_{i-1} and V_{i-2}) at the beginning of each output interval Δt subsequent to an enquiry of the conditions at the input. This is further explained through the waveforms shown in Figures (2), (3) and (4). Figure 2 shows the case when t_p, the pulse width of the dominant input I_i to a gate as measured between the two threshold points V_{T1} and V_{T2}, is small and V_{T2} follows V_{T1} before the output waveform V_i reaches the '0' level. As no storage time (TDD) considerations are involved, it may be seen that the response V_i is only a partial pulse; the settling time at the lower level of the output V_i (before the output starts to rise) is only due to the intrinsic charge in the switching circuit. Figure 3 shows the negative edge of the input I_i and the output V_i of a saturating gate at the '0' level initially. If the transition of the input from V_{T2} to V_{T1} has duration less than TDD, then the output does not change until a time TDD has elapsed after the point V_{T2}. A third case is illustrated by Figure 4 where the time from V_{T2} to V_{T1} (at the lower level of the input I_i) is larger than TDD. In this case the output rises at the end of TDD. If, however, the time

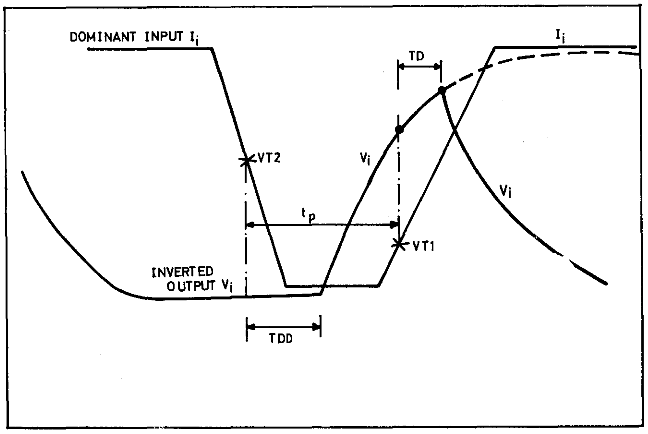


Figure 4—Effect of a short pulse width on the falling response

from V_{T1} to V_{T2} (at the higher level of the input I_i) is short such that the output V_i does not reach the '1' level, then the output continues to rise for a time TD from the point V_{T1} and then starts to fall.

These cases are obviously for inverting gates but the argument applies equally to gates without inversion at the output.

FEEDBACK LOOPS AND FLIP-FLOPS

A sequential circuit can be thought of as a combination of two simple NOR gates connected in a closed loop. The output of each NOR gate drives the input of the other. Actual sequential circuits do not in general behave ideally. Changes in voltages at the outputs of secondaries may not be simultaneous even though their inputs are; at high speeds even short interconnections may introduce significant delays. Unfortunately these various delays do not simply slow down the operation of the circuit, they may introduce functional errors and give rise to hazards. As a result of the inability to specify exactly when a given signal will actually make a transition from one state to the other, Szygenda, Rouse and Thompson[7] associate an ambiguity interval with each signal. The requirement of an ambiguity interval is due to the fact that gates of the same type could have different propagation times; hence the time delay of a gate is represented as a minimum value plus an ambiguity region. The approach in this paper, however, does not require the postulation of ambiguity intervals, since the propagation delays are accurately represented in the Macro-Model developed for each particular gate considered.

The simulation of feedback paths in a system is illustrated by considering Figure 7 in which module 8 forms a feedback loop with module 7. The scanning is started by giving an enabling initial condition to the feedback input of module 7. This may be done by considering the loop open at the beginning of the first time interval t_i; this provides an output for module 7 and hence for module 8. At the beginning of the next time interval, t_{i+1}, the feedback input is then made to take the value of the module 8 output thus giving the corresponding output value for module 7, and the entire output response is obtained by the usual iteration

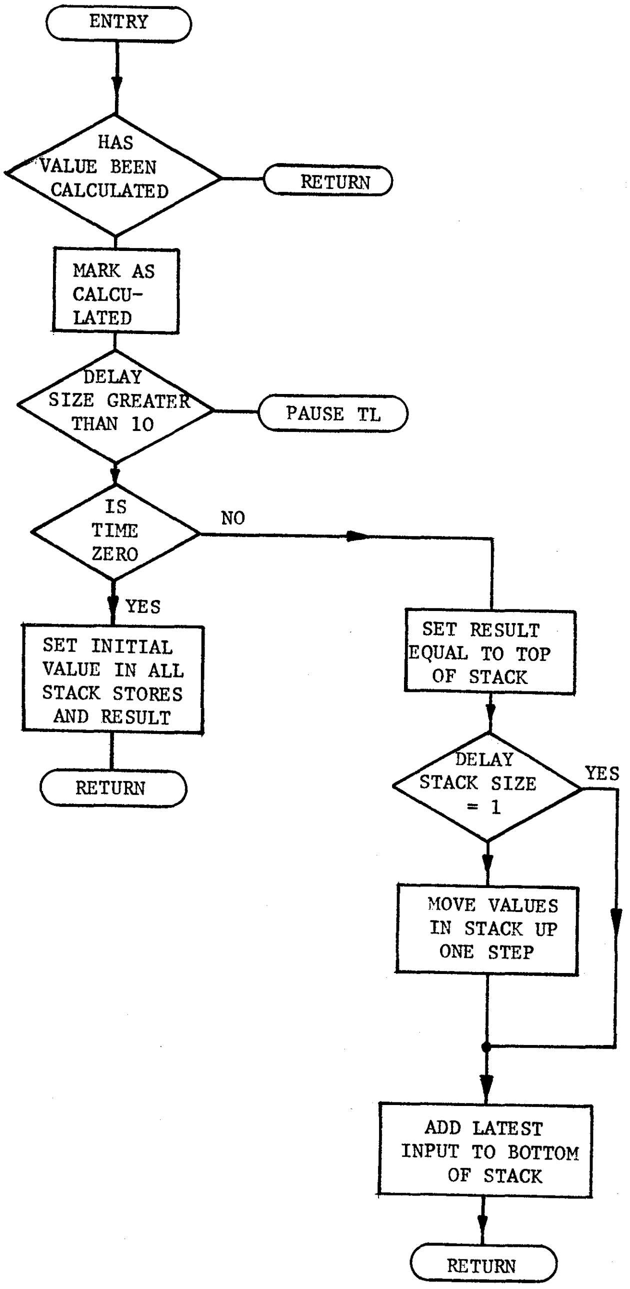

ENTRY
HAS VALUE BEEN CALCULATED
RETURN
MARK AS CALCU- LATED
DELAY SIZE GREATER THAN 10
PAUSE TL
IS TIME ZERO
NO
YES
SET INITIAL VALUE IN ALL STACK STORES AND RESULT
RETURN
SET RESULT EQUAL TO TOP OF STACK
DELAY STACK SIZE = 1
YES
MOVE VALUES IN STACK UP ONE STEP
ADD LATEST INPUT TO BOTTOM OF STACK
RETURN

Figure 5—Algorithm illustrating pure delay and unit-step delay

principle. This in effect may be interpreted as including a delay Δt in the feedback loop. It may be seen that the overall error in propagation time depends upon the ratio $\Delta t/nt_{pd}$ where t_{pd} is the individual delay of n modules constituting the feedback. This principle is easily extended to the simulation of sequential circuits e.g., flip-flops which can be considered as two cross-coupled inverting NOR gates. For inputs originating from a source with a pure delay, the values are those obtained during the previous iterations t_{i-d} determined from the respective values of the actual delay d as a function of time steps. This principle is illustrated in the flow chart of subroutine DELAY as shown in Figure 5.

MODEL IMPLEMENTATION

From the two different modes of handling time and processing events, i.e.,

(i) the next event formulation of a simulation model which is time variable and dependent on the events to be considered subsequently and
(ii) the fixed time step model,

the latter mode approach was chosen. A fixed time step model requires the user to specify a size for the time increment Δt to be used when the model is employed for simulation. In fixed time step simulation all response values are treated as if they all happened simultaneously at the upper end of the interval. Because of this, merely changing the size of the time increment to a large arbitrary value can radically alter the interrelationship of event occurrences.

It is noted that although a time step of any positive magnitude Δt allows the simulation to be executed, the meaning and validity of the results is related to the value of Δt. A very large arbitrary value of Δt may lead to inappropriate initial conditions. The conditions corresponding to a discontinuity or subregion may be left unscanned and the resulting response may have an incorrect subregional waveshape. Δt is chosen to be less than *TD* and *TDD* and is for example 1 or 2 *ns* for gates with switching times of the order of 10 *ns*. On the other hand, a fundamental problem associated with simulation is the time synchronization of model elements. This problem exists because the real network may perform certain phenomena simultaneously while the model must perform these phenomena in a sequential manner.

The simulator MILS (Macro-modelling of Integrated Logic Systems) is based on a "fixed-step" time-increment concept,[11] where a search is done for the next event after the current event has been met and passed.

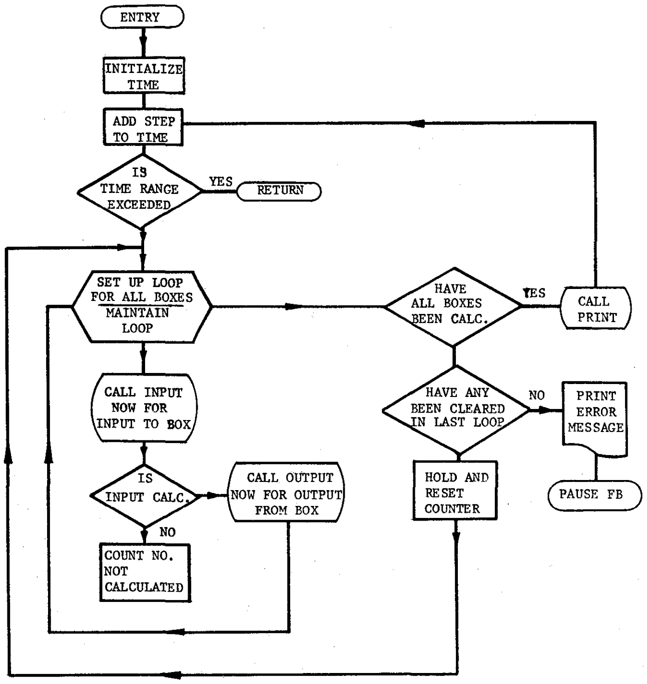


Figure 6—Algorithm illustrating the scanning procedure in simulated time

The actual simulation mechanism used, however, is a special case of this concept; in fact this mechanism can be considered as a special case of the "next-event" [11] principle, since all events do occur simultaneously at the ends of the time intervals and will be processed together until the next time is looked for. There is, however, a definite order in which events occur. This order is procedural[12] and is related to the logical interconnection and physical layout of the system. It is obvious that the card order in the simulation program may not be the same as the order of events occurring and in this respect the program description is non-procedural. In the simulation, each gate or circuit is treated as a box with a sequential number N as well as a type identification number. The description of the modules which may be of different kinds, are previously stored. Input functions, which may occur anywhere in the system, are designated "input-boxes" and included in the simulation with a sequential number. Each unit (or box) has response states at times t_i and t_{i-1}, the latter corresponding to the initial condition and, in general, the response states of boxes of similar or different type numbers need not have the same quantized value. To obtain the response of the system at a certain node, the corresponding discrete value of the input function is obtained and, while time stands still, the output of all the modules are evaluated by scanning in a sequential order. Owing to the topological configuration of a sys-

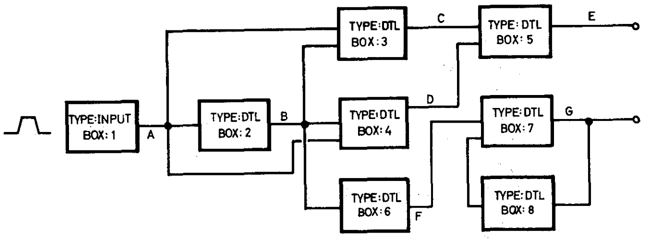


Figure 7—A circuit configuration which has DTL modules and includes a feedback loop; β_0 and F_t were taken as 10 and 500 MHZ respectively

tem, the outputs of some modules may not be available in a sequential scanning in which case the scanning of these modules is repeated with the help of pointers, until all the modules have functioned once. The operation then moves to the next interval, a new value of the input function is introduced, and the corresponding output value evaluated. The entire response is thus obtained by similar scanning. This procedure is illustrated in Figure 6.

Assuming identical box types, the output function V_{n,t_i} of the box n considered at time t_i is dependent on the status J_{n,t_i-1} of the output function. This status is determined from a knowledge of the position of the scanning time t_i with respect to the input function; and J_{n,t_i-1} has different values above V_{T1} and below V_{T2}. The m inputs of box n are first searched—not necessarily in a sequential order—to determine which input is dominant. This depends on the logic operation being performed. This resultant dominant input associated with input k is then designated I_{k,t_i} at t_i and then compared with the previous value I_{k,t_i-1} to determine the slope of the rising and falling input functions. If the j input to a box n is connected directly to an input-box, then the input value will be obtained from the appropriate specified input waveform at time t_i by a simple interpolation technique, but if the j input to box n is not from an input-box then the input value I_{k,t_i} will be extracted from the outputs of the m connecting boxes. If an input is from the output of the same box, this means that there is a feedback loop (see section 4). The previous value of the output function V_{n,t_i-1} is compared with the current value $V_{n,ti}$ to determine the initial conditions at each time interval. An analysis of the configuration shown in Figure 7 which has *DTL* modules and includes a feedback loop (i.e., between modules 7 and 8) was carried out by the simulation technique discussed, and the responses at nodes *A*, *B*, *C*, *D*, *E*, *F* are shown in Figure 8. The response at *G* remains as expected, high. A list of the symbols used in this section is given at the end of the paper.

SYSTEM IMPLEMENTATION

Speed of simulation is an important requirement but may be sacrificed if more accurate results are required through a reduction in Δt. Although an assembler language would result in a little faster execution and less storage requirement, FORTRAN was chosen since it would be easier to implement and is considerably more machine independent.[7] Its use is justified because the method itself is so efficient. The efficiency of this approach is illustrated in the storage and computing time required on Queen's University ICL 1907; 9 K for 300 logic modules, 40 types of logic gates, 10 input waveforms; and 21 mill seconds for the 8 gates configuration of Figure 7. FORTRAN is more compatible when dealing with numerical values in large arithmetic statements—as would be the case when computing the time response in numerical form. PLAN (ICL Machine-Coded Language) or PL/1 has desirable features which could be easily added to support the FORTRAN IV system only in the part of the program not dealing

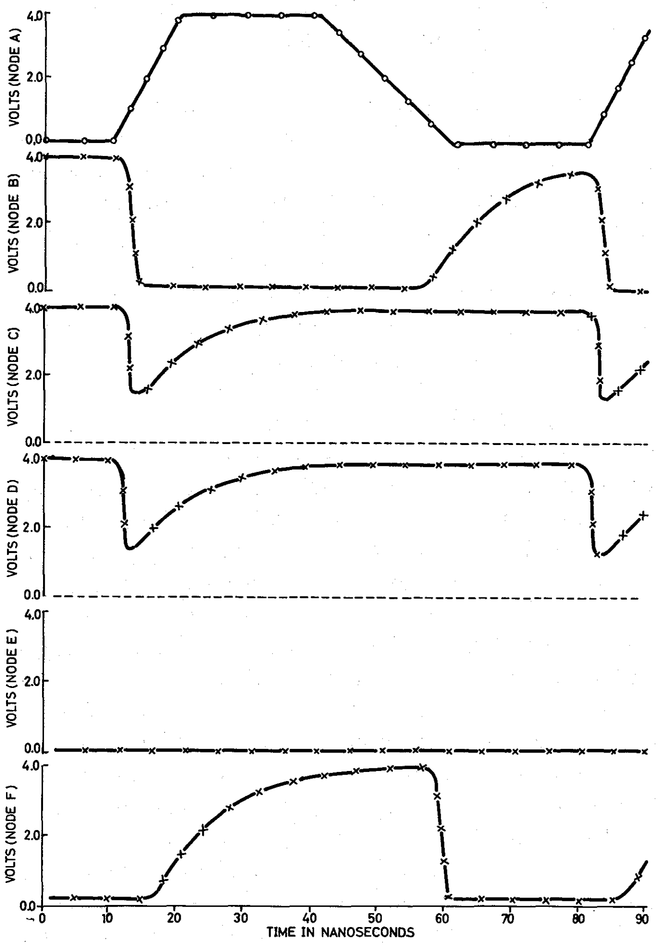


Figure 8

with numerical values especially in the STORE part, hence increasing the efficiency.

However, FORTRAN programs can grow excessively large for two reasons: firstly that they contain a large number of instructions, and secondly that they require a large amount of storage, usually for arrays. Programs which are large for the first reason can usually be overlaid. MILS provides a way of treating the storage in a similar fashion. An area in core can be used to hold the part of a larger storage area that is required at a given time, selected from the larger area held on disc storage. When the part is no longer required it can be written away to the disc storage and the next part can be brought down into core store previously occupied by the first segment. There is no restriction on the order in which sections of storage are accessed. The storage described in this paper is scratched space, i.e., any information stored on it is lost when MILS is terminated. The system uses files on magnetic disc devices to store the information. Files can be on either EDS or FDS (exchangeable disc storage or fixed disc storage) scratch files. A maximum of eight files can be declared at any one time. A file must be declared by a CALL to a subroutine which declares a scratch file of length K elements. One element is required for each real or integer array element. Two elements are required for each complex or double-precision array element; another parameter is for the channel number by which the file is to be referred and must be an integer; a third parameter is for EDS or FDS. The area on the disc is treated as a string of elements numbered from one upwards. It is required to keep track of the start positions of each of its arrays on the disc by means of pointers. This is illustrated by the two subroutines:

```
    CALL STACK (C, K, A)
    CALL ACCESS (C, K, A)
```

where C in the channel number, K is the element number of the first element to be transferred and must be a variable as the value of K will be altered by any of the routines and A is the array name.

X and Y are array elements declared as A(I, J - - - -), A(L, M - - - -). The values given to all dimensions of the array element Y must be greater than or equal to those of X.

Each of these subroutines writes to or reads from the file with channel number C starting at element K. K must be a variable and have a value greater than or equal to 1.

CONCLUSIONS

This formulation technique of an integrated logic module permits the analysis of large networks, consisting of hundreds of logic gates, with complex situations and allows a more accurate prediction of transient performance and propagation delay. The method can be used for studying statistical variations of delays and switching times without employing regressional analysis. The individual elements of the logic module could also be modified for worst case analysis without affecting its macro-appearance. Further, the approach lends itself to predicting the effects of multiple reflections and crosstalk in a system with a given number of interconnecting paths and tap-offs. An analysis of these effects forms an important and interesting part of the overall system simulation. While a large amount of work has been done[3] to simulate these effects, in view of the space and time available, a discussion of this aspect will be outside the scope of this paper.

LIST OF SYMBOLS

I_{k,t_i}	Dominant input function to a box n at time t_i determined from previous boxes' outputs I_i for simplicity
$I_{k,t_{i-1}}$	Dominant input function to a box n at time t_{i-1} $(t_{i-1}=t_i-\Delta t)$ determined from previous boxes' outputs I_{i-1} for simplicity
$J_{n,t_{i-1}}$	Status of the output function at time t_{i-1}
j	Input number considered
k	Dominant input box number
m	Maximum number of inputs to a box
N	The sequential number given to a box
n	The box number being considered excluding all the preceding input boxes $\mid x \mid$ $\mid N-x \mid$
t_i	Current time considered
t_{i-1}	Previous time considered $(t_{i-1}=t_i-\Delta t)$
t_{i-d}	Previous iteration time determined from the respective values of the actual delay d as a function of time steps $(t_{i-d}=t_i-d)$
TD	Delay time above V_{T1}
TDD	Storage time below V_{T2}
V_{n,t_i}	Output function at t_i for the box considered n V_i for simplicity
$V_{n,t_{i-1}}$	Output function at t_{i-1} for the box considered n V_{i-1} for simplicity
V_{T1}	Threshold point for a leading input edge
V_{T2}	Threshold point for a trailing input edge

ACKNOWLEDGMENTS

The author wishes to record his thanks to Professor W. D. Ryan for his inspiration in the development of the macro-modelling technique, to Mr. S. Q. Hossain for his initiation to the interconnection problem, to

Mr. Sam Hogg and the Staff of the University's Computer Laboratory for useful discussions in the design implementation of the simulator and to my wife, Elfriede, who typed the manuscript.

REFERENCES

1 J A NARUD
Computer-aided analysis and artwork generation for integrated circuits
IEEE International Convention New York 1970

2 J A NARUD M L DERTOUZOS G P JESSEL
Computer-aided design for integrated circuits
IEEE International Symposium on Circuit Theory Florida December 1968

3 N B RABBAT
Macro-Modelling and transient simulation in large integrated digital systems
PhD thesis The Queen's University of Belfast October 1971

4 N B RABBAT W D RYAN S Q HOSSAIN
Computer modelling of bipolar logic gates
Electronics Letters Vol 7 pp 8-10 1971

5 D KOEHLER
Computer modelling of logic modules under consideration of delay and waveshaping
Proceedings of the IEEE Letters Vol 57 pp 1294-1296 1969

6 G G HAYS
Computer aided design: simulation of digital design logic
IEEE Transactions on Computers Vol C-18 pp 1-10 1969

7 S A SZYGENDA D M ROUSE E W THOMPSON
A model and implementation of a universal time delay simulator for large digital nets
Spring Joint Computer Conference AFIPS Proceedings Vol 36 pp 207-216 1970

8 D K LYNN C S MEYER D J HAMILTON
Analysis and design of integrated circuits
McGraw-Hill Book Co Inc New York 1967

9 N B RABBAT W D RYAN S Q HOSSAIN
Transient analysis of modified TTL gate
Electronics Letters Vol 7 pp 22-24 1971

10 N B RABBAT W D RYAN S Q HOSSAIN
Macro-modelling and transient simulation in integrated digital systems
To be published International Computer-aided design Conference University of Southampton England April 1972

11 G W EVANS G F WALLACE G L SUTHERLAND
Simulation using digital computers
Prentice-Hall 1967

12 B J KARAFIN
The new block diagram compiler for simulation of sampled data systems
Spring Joint Computer Conference AFIPS Proceedings Vol 27 pp 53-62 1965

Functions for improving diagnostic resolution in an LSI environment*

by MADHUKUMAR A. MEHTA

GTE Automatic Electric Laboratories Incorporated
Northlake, Illinois

and

HENRY P. MESSINGER

Illinois Institute of Technology
Chicago, Illinois

and

WILLIAM B. SMITH

Bell Telephone Laboratories Incorporated
Holmdel, New Jersey

INTRODUCTION

LSI implementation of digital circuitry opens the door to the consideration of dramatically new approaches to the design of system fault diagnosis.[1] New constraints have been added, such as the difficulty of inserting test access points internal to large pieces of circuitry. At the same time, failure modes seem to be changing with bonding lead failures increasing in importance.[2,3] This paper presents an approach that leans heavily on the assumption that adding additional logic to a circuit is of little consequence, whereas it is important to reduce the access provided for testing capability. As the practicality of the proposed approach has not been examined in detail, the concept is primiarly presented to stimulate further study into the special problems and opportunities involved in diagnosis of LSI systems.

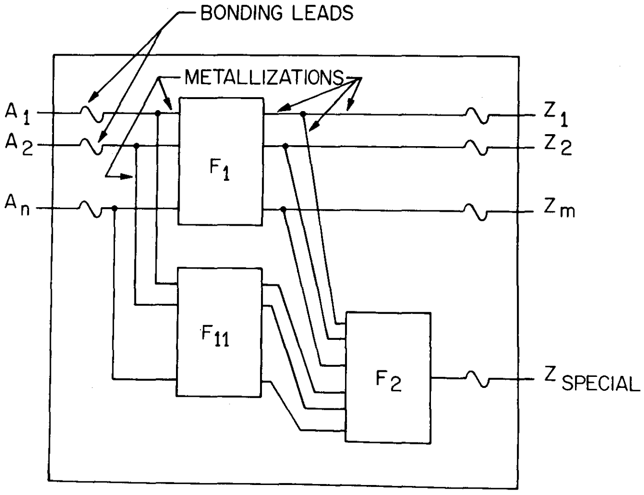
The approach discussed here is to incorporate on each least replaceable unit (LRU) a special combinational test circuit that will identify, by observation of the output alone, any stuck at "0" or "1" input failures to the particular input involved. Such a test circuit is termed an "Ambiguity Resolver" (AR) function, and proof of its existence for any number of inputs will be presented. The identification of a failure to a particular input in general actually only isolates the failure to the bus that is connected to that input because of the propagating tendency of failures on a bus. Thus, all LRU's connected to the bus implicated are equal candidates to have failures on their bus connections. A capability such as provided by AR functions would obviously be of little use in a situation in which buses tend to be common to a group of LRU's. However, when the observable points are restricted to the functional outputs, conventional methods usually do not provide resolution of input/output faults to a unique bus. This feature is conveniently provided by the usage of AR functions.

Let us define some of the terms and clarify the notations used in this article.

DEFINITIONS AND NOTATIONS

Observable Points: points at which the outcome of an applied test procedure can be observed.

* Parts of this paper are excerpts from Dr. Mehta's Ph.D. Thesis. Dr. Messinger of Illinois Institute of Technology served as an academic adviser during the course of the thesis, while Dr. Smith of Bell Telephone Laboratories acted as a thesis adviser. Dr. Mehta is with GTE Automatic Electric Laboratories which also supported the thesis.



F1 : THE BASIC FUNCTION

F11: THE DUPLICATED FUNCTION

F2: A COMPARE FUNCTION

Figure 1—One way to use redundancy

Failure Exclusive Points: a set of points only one of which is affected by any one assumed failure.

Independent Inputs: two inputs are independent iff (if and only if) the logic state of either can be changed without affecting the logic state of the other.

Sensitized Path: a path between a point in a combinational circuit and the output is sensitized if a change of logic conditions at that point results in a change of logic conditions at the output while other inputs are not changed.

Output Vector: the ordered set of (binary) outputs that result from the application of a test procedure (a sequence of tests). The output vector corresponding to no-fault conditions is called the *normal output vector* and is denoted by Σ_n; also Σ_f denotes the output vector for the same test procedure when fault f is present.

Complementary Failures: Stuck-at-0 (s-a-0) and Stuck-at-1 (s-a-1) failures at a particular point (input or output) are called *complementary.*

Least Replaceable Unit (LRU): the smallest subsystem which will be completely replaced if a fault is located at the terminals of, or inside, the subsystem.

Capital letters of the English alphabet are used to denote the inputs and the outputs of a circuit. Usually the early letters (A, B, C) are used for the inputs while the later letters are reserved for the outputs (Z, Y, X, W). Exceptions in these notations are made when dealing with networks to avoid duplicate labeling for connections (buses). Subscripts are used to identify the inputs or the outputs. A distinction is made between the inputs to the circuit (A_i) and the leads which carry these inputs to the circuit. Small letters with identical subscripts (a_i) designate the corresponding leads for any input. Similar notations are used to distinguish the outputs and the output leads. A fault at the input lead is fed back to the input (output of the driving gate) only if it is a propagating fault (i.e., if it affects the entire bus). A lead a_i with a s-a-0 fault, in tables and figures, is denoted by symbol a_i^0; a_i^1 similarly denotes the same lead with a s-a-1 fault.

AMBIGUITY RESOLVER (AR) FUNCTION

Since LSI implementation makes redundant logic economic to use, we examine the possible ways of using redundancy (internal to a LSI circuit) to simplify fault diagnosis of digital systems.

One of the ways to use the redundancy is the conventional approach of duplicate and compare (Figure 1). Two disadvantages prevent this scheme from being attractive. First, the required redundancy is excessive; it takes more than double the circuitry to implement this approach. Second, the approach simplifies the diagnosis of only the internal faults. For the diagnosis of the faults at the bonding leads one has to still resort to other approaches. Unfortunately, a predominant failure mode in systems composed of LSI, that have been in systems which are in operation for awhile, is due to bond failures.[2,3] Therefore, the approach of

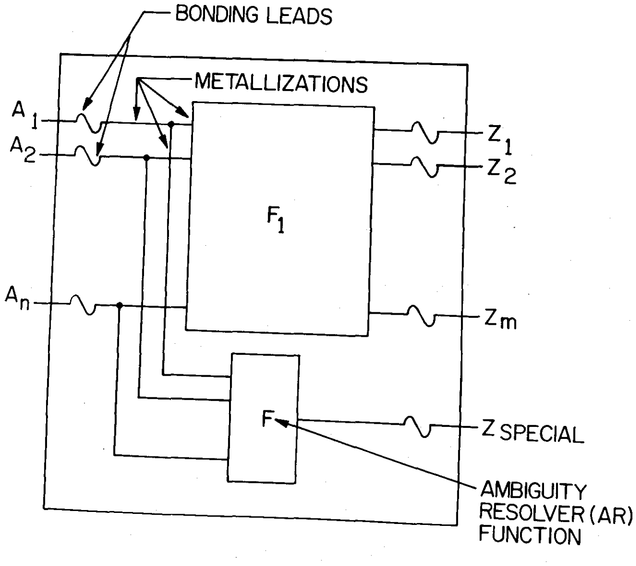


Figure 2—Another way.to use redundancy

Figure 1 does not represent an efficient usage of redundancy.

The proposed approach (Figure 2) consists of a specialized single output test function (F) for simplifying the diagnosis of input faults. These Ambiguity Resolver (AR) functions are combinational logic functions which have the property that input/output faults can be resolved to a unique bus through the observation of the output alone. We will show that such functions exist and that they are simple in comparison to the normal function (F_1). AR functions simplify the fault diagnosis of digital systems implemented with LSI circuits because:

(1) A predominant failure mode in operating IC's is bond failures which result in input/output faults. The AR functions simplify the diagnosis of only the input faults; however, the output faults will become input faults of the driven LRU's and therefore will be diagnosed by their AR functions. It is reasonable to assume that the faults of the unused outputs do not affect the system performance.

(2) AR functions permit resolution of input/output faults to a unique bus. Such resolution is not always available by the use of conventional test procedures when the observable points are restricted to the functional outputs.

(3) Only one point need be observed for diagnosis.

(4) Since the AR function (F) is unrelated to the basic function F_1, its design is completely independent; for example, F_1 may be a sequential function, yet F can be a simple combinational function. Due to this independence, often diagnosis using AR functions requires fewer tests in comparison to a conventional test procedure.

Although the AR approach simplifies fault diagnosis of LSI implemented systems, it is certainly not the complete answer. For example, the AR approach requires testing, and thus does not provide an on-line detection of faults. Also AR function provides neither fault detection nor diagnosis of faults within an LRU. Therefore, in any practical application, the AR approach would probably be augmented with other, more conventional methods.

Since the AR function can be independently designed, in the following sections, we will disregard the basic function. We will concern ourselves only with single output combinational functions which have the AR property. We will assume:

1. Only input/output faults.
2. Only s-a-1, s-a-0 faults.
3. One fault at a time.
4. Access to only the inputs of the function. That is, only the inputs can be controlled externally.
5. Output is the only observable point, i.e., the performance of the function, during the test procedure, can be observed only at the output.

TABLE I—Outcomes of a Test for Complementary Failures at the Input a_k and their Implications

Test	Input Conditions A_1-----A_n	Normal Output	Possibility	a_k^0	a_k^1	Implications for the test
t_i	$\alpha_{1i}\alpha_{2i}$---α_{ni}	σ_i	1	σ_i	σ_i	$\alpha_{ki} = 0$ or 1 fault insensitive
			2	σ_i	$\bar{\sigma}_i$	$\alpha_{ki} = 0$
			3	$\bar{\sigma}_i$	σ_i	$\alpha_{ki} = 1$
			4	$\bar{\sigma}_i$	$\bar{\sigma}_i$	Not realizable

In order to clarify some of the concepts presented here, Section 8 presents an example of the use of AR functions along with a conventional scheme for comparison.

COMPLEMENTARY FAILURES

Understanding the constraints that are imposed on circuits being tested is required before we can go into a discussion of the existence of AR functions. An important constraint is on the allowable outputs for complementary input failures.

Consider a test t_i that applies the input $\alpha_{1i}\alpha_{2i} \ldots \alpha_{ni}$ to a circuit, and consider a particular input lead a_k (see Table I). If the output of the circuit is σ_i under the test t_i, then the output can be either σ_i or $\bar{\sigma}_i$ for a_k s-a-0 and similarly for a_k s-a-1. If the output is σ_i for a_k both s-a-0 and s-a-1, then the output is fault insensitive to a_k. If the output is $\bar{\sigma}_i$ for a_k s-a-0 and σ_i for a_k s-a-1, then it is clear that a_k must have a "1" input for this test t_i. A similar story is true for output σ_i for a_k s-a-0 and $\bar{\sigma}_i$ for a_k s-a-1 (a_k has a "0" input). However, it is not possible to have an output $\bar{\sigma}_i$ for both kinds of failures because this would imply that $\bar{\sigma}_i$ is always the output for a_k with either 0 or 1 input, which contradicts the original assumption. Theorems 1 and 2 state the results just given, which are proved elsewhere in detail.[7] They are given below because the results will be used later.

Theorem 1:

The outcome of a test t_i cannot be different from normal case under both a s-a-0 and a s-a-1 failure at the same input.

Theorem 2:

If the outcome of the test t_i is different from the normal case for the fault f_i, then the outcome of the test t_i for the complementary failure is identical to the normal case.

Finally, each detectable failure must produce a non-normal output for at least one of the tests. For that test, by Theorem 2, the complementary failure must have normal output. Therefore', the resulting output vectors for complementary failures cannot be the same. Thus, although this is not necessary for localization of input faults to a unique bus, complementary failures of AR function will always be distinguishable. Having established this, we are now in a position to formally define the AR function.

DEFINITION AND PROPERTIES OF AR FUNCTIONS

Definition: An n-input AR function is a combinational function of n variables with a single output with the following properties:

1. All faults are detectable, i.e., there exists a test procedure T containing p tests such that $1 \leq p \leq 2^n$ and $\Sigma_n \neq \Sigma_{f_i}$ for any fault f_i.
2. All faults produce unique output vectors for the test procedure T, i.e., $\Sigma_{f_i} = \Sigma_{f_j}$ iff $f_i = f_j$.

Any such set of p tests will be referred to as *Test Pattern* of the AR function (under the fault assumptions made earlier).

Some elementary properties are easily derived from the basic definition of the AR function. Clearly, the output faults yield either an all 0's (s-a-0) or an all 1's (s-a-1) output vector. Also, the normal output vector for the test pattern has to be non-zero and non-all 1's since output faults are to produce non-normal output vectors. Finally, each input fault must yield a unique, non-zero and non-all 1's output vector.

Theorem 3:

If the permissible failure modes include complementary faults at the inputs and at the outputs, then the complement of the normal output vector for an AR function is different from the resulting output vectors for all permissible failures.

Proof:

Since the normal output vector is non-zero, non-all 1's, the complement of the normal output vector ($\Sigma_{\bar{n}}$) is also non-zero, non-all 1's. Thus $\Sigma_{\bar{n}}$ is different from the resulting output vectors for the output faults.

Next assume that the resulting output vector for an input fault f_i is identical to $\Sigma_{\bar{n}}$. This implies that the outcome for each test t_j $(1 \leq j \leq p)$ for the fault f_i is different from the normal case. Now consider the complementary failure $\bar{f}_i$. By Theorem 1, for any test t_j the outcome for both failures at the same input cannot have different performance from the normal. Therefore (by Theorem 2), the outcome for each test t_j $(1 \leq j \leq p)$ for the fault $\bar{f}_i$ must be identical to the normal case. Thus the resulting output vector for the fault $\bar{f}_i$ for the test pattern T will be identical to the normal case. But this is a contradiction since the fault $\bar{f}_i$ during the test pattern T must produce a unique output vector *different* from the normal case. Thus the resulting output vector of an input fault of an AR function also cannot be identical to $\Sigma_{\bar{n}}$. (Q.E.D.)

Next we examine other properties of AR functions which will permit synthesis of new AR functions from

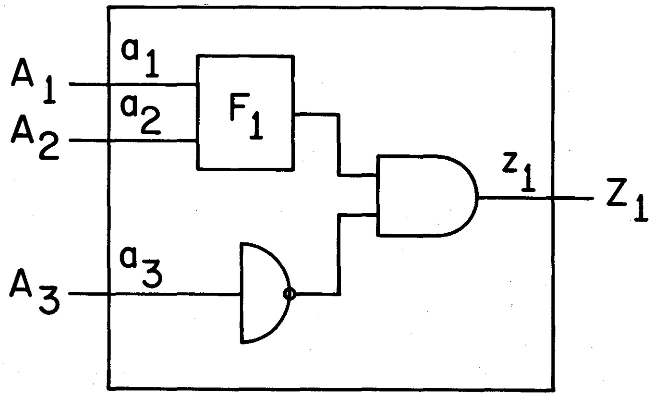


Figure 3a—Not conformal because faults at a_3 violate condition 1

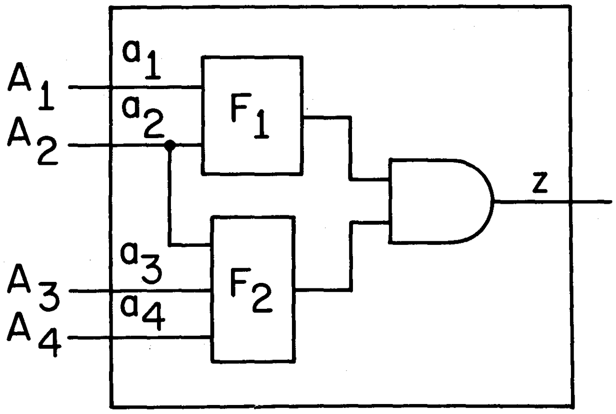


Figure 3b—Not conformal because faults at a_2 violate condition 1

the available AR functions. It will be shown that the resulting functions have the AR property when certain structural constraints are observed. The concept of conformal structure, introduced next, provides one convenient way to produce new AR functions. It should be noted, however, that although conformal structures, as defined below, have the AR property, structures having the AR property need not be conformal.

Definition:

When a one output combinational function is generated by the use of AR functions and logic gates, the resulting structure is *conformal* iff

1. Each input failure of the resulting function results in a permissible input failure at one, and only one, of the AR functions,
2. No two input failures of the resulting function result in an identical input failure condition at any AR function,

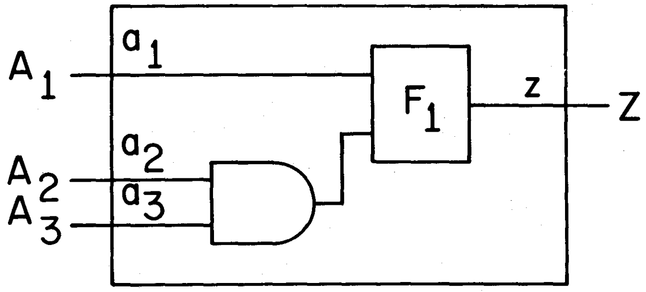


Figure 3c—Not conformal because s-a-0 faults at a_2 and a_3 violate condition 2

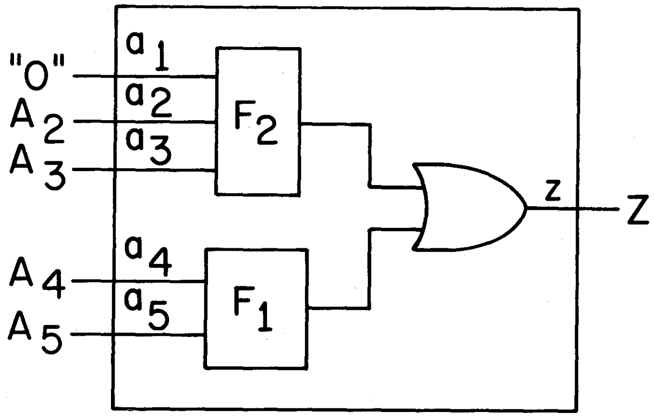


Figure 3d—Not conformal because the test pattern for F_2 may not be possible (condition 3)

3. Application of the test pattern for each AR function is possible by control of the inputs of the resulting function, and
4. For the output of each AR function, a sensitized path to the output of the resulting function exists.

Figure 3, where F_1 and F_2 are AR functions, illustrates a number of structures which are not conformal for one or more reasons.

The motivation for the use of various conditions in the above definition can be intuitively explained at this point. Condition 1 insures that each input fault can be made distinguishable from the other faults by the test pattern of one of the AR functions. The second part of the condition ("only one") prevents one input fault

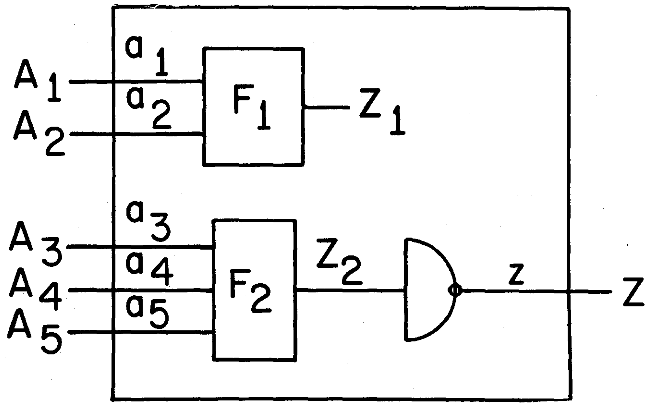


Figure 3e—Not conformal because the output Z_1 violates condition 4

from affecting more than one AR function. The resulting output vectors of each AR function need a sensitized path to the final output to be observable under normal, as well as under various failure conditions. One of the ways to provide such a path is to maintain a unique combination of the inputs at the remaining AR functions (which provides a sensitized path) during the testing of each AR function. Following this procedure, in addition, permits distinguishability of the input faults of one AR function from the input faults of other AR functions as we will see. If an input fault of the resulting structure affected more than one AR function, this procedure may not be possible in all cases. Condition 2 prevents multiple input failures from having identical output vectors, thereby permitting the required localization. Condition 3 assures the ability to apply test patterns to various AR functions, and the condition 4 makes the resulting output vector available at the final output of the resulting structure.

Having established some structural constraints and some intuitive basis for them, we proceed to prove the AR property of conformal structures.

Theorem 4:

A conformal structure implemented with AR functions and logic gates results in an AR function.

Proof:

Let $F_1, F_2, \cdots F_n$ be the n AR functions used in the conformal structure. We will refer to these functions as component AR functions. Let Z_i be the output of the function F_i and let Z be the output of the resulting function F with conformal structure.

To prove the AR property of the resulting conformal structure, we will show the existence of a test procedure which provides the distinguishability of the input/output faults of F.

As the structure is conformal, each input fault of the resulting structure results in a permissible input fault at one (and only one) AR function. Thus the set of input faults of the resulting structure is a subset of the input faults of the component AR functions. Also due to the second condition of conformality, these input faults of F produce unique input faults at the component AR functions. Thus if the input faults of the component AR functions are distinguishable, then the input faults of the resulting function are also distinguishable.

Consider a test procedure wherein we apply the test pattern to each component AR function, one after another, by control of some of the inputs while maintaining the remaining inputs at some logic conditions so as to provide a sensitized path to the output Z throughout the test pattern. The conformality of the structure accounts for the feasibility of such a test procedure. We will show that this test procedure provides the required distinguishability.

To show the distinguishability, we will follow these steps. First we will show that input faults of any F_i are distinguishable from the other input faults of the same AR function. Next we will show that these faults are distinguishable from the faults at the output Z. Then we will show the distinguishability of these faults from the input faults of the other AR functions. Note that the distinguishability is symmetric, i.e., if a is distinguishable from b, then b is distinguishable from a. Therefore, the distinguishability of the output (Z) faults from the input faults of the resulting structure is easily established. Finally, we will show the distinguishability of the output faults from each other.

During the test procedure, the test pattern T_i is applied to F_i, and a sensitized path to the final output for the output Z_i is provided. As F_i is an AR function, its input faults produce unique non-normal output vectors at Z_i for the test pattern T_i which due to the availability of the sensitized path are transmitted to the output Z. (The normal output at Z may be the same as, or could be the complement of, the output at Z_i during the test pattern T_i as sensitization does not guarantee the polarity of the output. But as the uniqueness is what we are concerned with, and as uniqueness is preserved under complementation, we do not worry about the polarity.) Note that due to the conformality of the structure during the test pattern T_i only permissible faults (s-a-0 or s-a-1) exist at the inputs of the function F_i. Further, as we consider only a single failure, when we are considering the distinguishability of the two input faults of F_i, the path sensitization can be assumed to be failure free. Thus the input faults of F_i produce unique output vectors at the output Z_i, which in turn, produce unique output vectors at the output Z. Therefore, the input faults of F_i are distinguishable from the other input faults of the same AR function F_i, when test pattern T_i is applied as a part of the proposed test procedure by observing Z. Also, as the resulting output vector at the output Z for each input fault is different from normal, the input faults are detectable.

Now consider the distinguishability of the input faults of the function F_i from the output faults at the output Z. Observe that when the test pattern T_i is applied, as a part of the test procedure, the observed output vector for the faults at the output Z will be either all

0's or all 1's. While the input fault of the function F_i results in a non-normal, non-zero, non all 1's output vector at the output Z_i, and will be transmitted to the output Z due to availability of a sensitized path. Also as all the properties (non-normal, non-zero, non all 1's) of the output vector are preserved even under complementation, the resulting output vectors at the output Z, for the input faults of the AR function F_i, are non-normal, non-zero and non all 1's. Therefore, the input faults of F_i are distinguishable from the faults at the output Z when the test pattern T_i is applied.

Now consider the distinguishability of the input faults of one AR function, say F_i, from those at the inputs of the other AR functions, say F_j where $i \neq j$. The two AR functions, F_i and F_j, do have independent inputs* due to condition 1 of the conformal structure. Therefore, during the testing of F_i by the test pattern T_i, to provide a sensitized path a single combination of the inputs of F_j can be used throughout the test pattern T_i. The input faults of the function F_j would result in one of four cases during the test pattern T_i.

1. It does not affect the final output.
2. It makes the final output a complement of what should be (changes the polarity of sensitization), i.e., fails all tests in T_i.
3. Makes the final output a logic 0 for all tests.
4. Makes the final output a logic 1 for all tests.

In the first case, the resulting output is normal; in the second case, it is always the complement of the normal; in the third and the fourth cases it is at a permanent logic condition for the test pattern T_i. Since the input faults of F_i, on the other hand, result in unique, non-normal, non-zero, non all 1's output vectors, they are distinguishable from the input faults of F_j in cases 1, 3, and 4. Further, by Theorem 3, the complement of the normal output vector cannot be the same as any output vector for the input faults. Therefore, the input faults of the functions F_i and F_j are distinguishable.

As we repeat the procedure for each AR function, each input fault of every component AR function will be distinguishable from other input faults and from the output faults at Z.

Finally, as the property of distinguishability is symmetric, the output faults are distinguishable from the input faults. Also as the output faults produce an all 0's or all 1's output vector and as normal output vector has a logic 0 and a logic 1 element, the output faults are detectable and distinguishable from one another.

Therefore, as the test procedure considered, provides unique, non-normal, output vectors for various faults, the resulting conformal structure has the AR property and the test procedure is its test pattern. (Q.E.D.)

It becomes apparent that the formulation of conformal structure provides us with a great flexibility in generating new AR functions. Of these, three are worthy of specific mention.

Corollary 1: The *complement* of an AR function is an AR function.

Corollary 2: The AND function of AR functions is an AR function if the resulting structure is conformal.

Corollary 3: The OR function of AR functions is an AR function if the resulting structure is conformal.

EXISTENCE OF A GENERAL N-INPUT AR FUNCTION

There are a number of ways by which we can prove the existence of an n-input AR function, for any n. For instance, it is easy to show that a parity network* with any number of inputs has the AR property. However, for three or more inputs other functions, apparently less complex, also exhibit the AR property. We, therefore, will create simple two- and three-input AR functions and use them in a constructional proof for proving the existence of a general AR function. This proof is preferred because it also provides a scheme for generating relatively simple AR functions.

Existence of a 2-input AR function

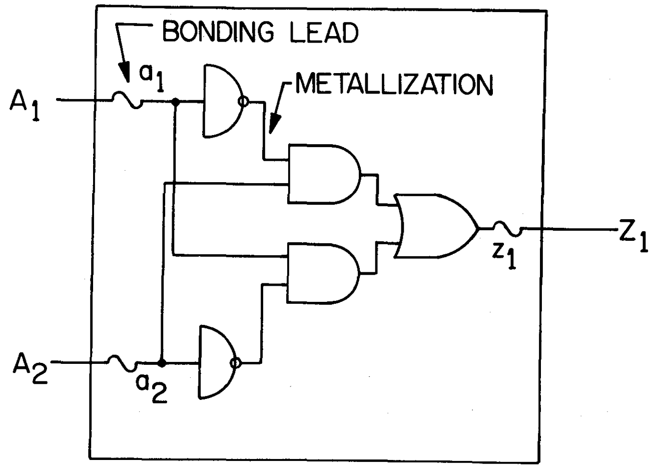
Figure 4 shows a two input exclusive OR function. Also shown are the resulting functions for various faults. Note the uniqueness of the resulting function for each fault. As the resulting function for each fault is unique and non-normal the exclusive OR function is an AR function. Table II shows a test pattern for the AR function. That this is a test pattern can be shown by generating the resulting output vectors for various conditions and showing their uniqueness.

Existence of a 3-input AR function

Figure 5 shows a three input function. Also shown are the resulting functions for various faults. As the result-

* An input fault of F only affects one of the F_i's.

* A single output network using exclusive OR gates.



$$Z_1\ (\text{NORMAL}) = a_1\,\bar{a}_2 + \bar{a}_1\,a_2$$

$$\begin{array}{llll} Z_1(z_1{}^0) & = 0 & Z_1(a_1{}^1) & = \bar{a}_2 \\ Z_1(z_1{}^1) & = 1 & Z_1(a_2{}^0) & = a_1 \\ Z_1(a_1{}^0) & = a_2 & Z_1(a_2{}^1) & = \bar{a}_1 \end{array}$$

Figure 4—A 2-Input AR function

ing function for each fault is unique and non-normal the function has the AR property. Note that the function of Figure 5 is simple compared to a three input parity network. Table III shows a test pattern for the AR function of Figure 5. That this is a test pattern can be shown by generating the resulting output vectors for various conditions.

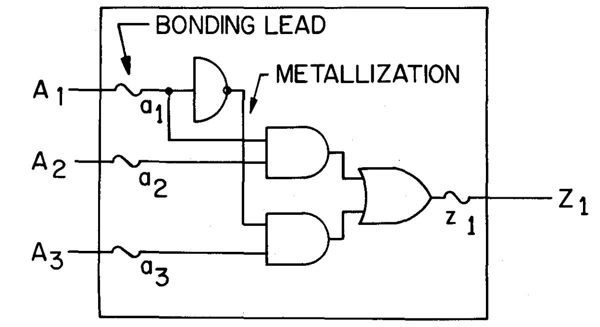
Having shown the existence of 2- and 3-input AR functions we now show the existence of a general n-input AR function for $n \geq 2$.

Theorem 5:

There exists an n-input AR function for any finite $n \geq 2$.

TABLE II—A Test Pattern for the 2-Input AR Function of Figure 4

Test No.	Inputs		Normal Output
	A_1	A_2	
1	0	0	0
2	0	1	1
3	1	0	1



$$Z_1\ (\text{NORMAL}) = a_1 a_2 + \bar{a}_1 a_3$$

$$\begin{array}{llll} Z_1(z_1{}^0) & = 0 & Z_1(a_2{}^0) & = \bar{a}_1 a_3 \\ Z_1(z_1{}^1) & = 1 & Z_1(a_2{}^1) & = a_1 + \bar{a}_1 a_3 = a_1 + a_3 \\ Z_1(a_1{}^0) & = a_3 & Z_1(a_3{}^0) & = a_1 a_2 \\ Z_1(a_1{}^1) & = a_2 & Z_1(a_3{}^1) & = a_1 a_2 + \bar{a}_1 = a_2 + \bar{a}_1 \end{array}$$

Figure 5—A 3-Input AR function

Proof:

We have already shown the existence of 2-input and 3-input AR functions. Consider any integer $n>3$, then it is always possible to find integers q and r ($2 \leq q \leq n/2$ and $0 \leq r \leq 1$) such that $n = 2q + 3r$. In other words, for all even integers, we have $r=0$, and $q=(n/2)$ while for odd integers $r=1$ and $q=[(n-3)/2]$.

Since the OR function of AR functions is an AR function by Theorem 4 (Corollary 3), we can generate an n-input AR function for $n>3$ by generating an OR function of q 2-input AR functions and r 3-input AR functions with conformal structure. Therefore, n-input AR function exists for any finite $n \geq 2$. (Q.E.D.)

THE CONVERGENCE PROPERTY

In the previous sections, we examined few techniques to generate new AR functions. The functions, thus

TABLE III—A Test Pattern for the 3-Input AR Function of Figure 5

Test No.	Inputs			Normal Output
	A_1	A_2	A_3	
1	0	1	0	0
2	0	0	1	1
3	1	0	1	0
4	1	1	0	1

generated, had the property of failure localization to a single bus for the input/output faults of the resulting function. To that extent, these new functions can be used internal to the LRU to provide the failure localization capability. If we provide an AR function in each LRU, the special output (the output of the AR function) of each LRU must be made observable. Does the possibility of reducing the total number of observable points exist? To answer this, we consider if convergence of AR functions by an AR function preserves the failure localization property.

First, let us illustrate what we mean by a converging connection. Let $F_1, F_2, \cdots, F_n$ be n AR functions. Further, let each F_i have m_i inputs and one output. Also let F_c be an AR function with n inputs. Now consider a configuration, shown in Figure 6, where the output Z_i of each F_i is connected to the ith input of F_c. Let Z be the output of F_c. The resulting structure will be termed a converging connection.

It can be shown[7] that when the inputs to a converging connection are independent and failure exclusive, the input/output faults of each component AR function (F_i's or F_c) are locatable by observing Z alone. Thus a test procedure exists which produces unique output vectors at the final output Z for each of the faults at the inputs (a_{ij}; $1 \leq i \leq n$, $1 \leq j \leq m_i$) or at the outputs (Z_i's or Z_c). Further, it can be shown that the length of such a test procedure does not exceed the sum of the tests in the test patterns of the component AR functions. The detailed proof of this result, which follows a similar line of reasoning as that of Theorem 4, is rather lengthy and is, therefore, not included.

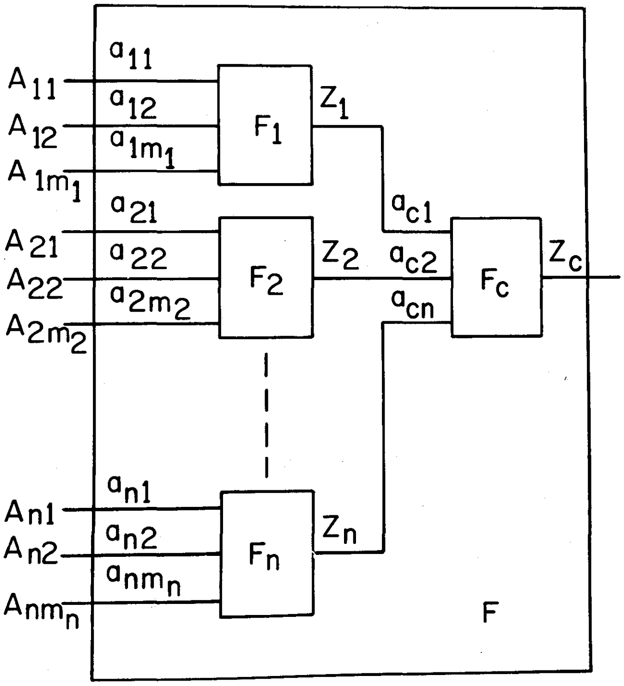
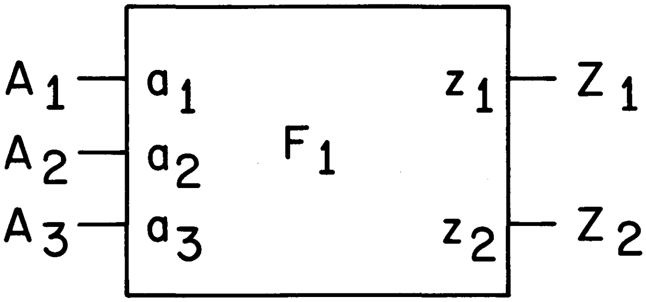


Figure 6—A convergent structure



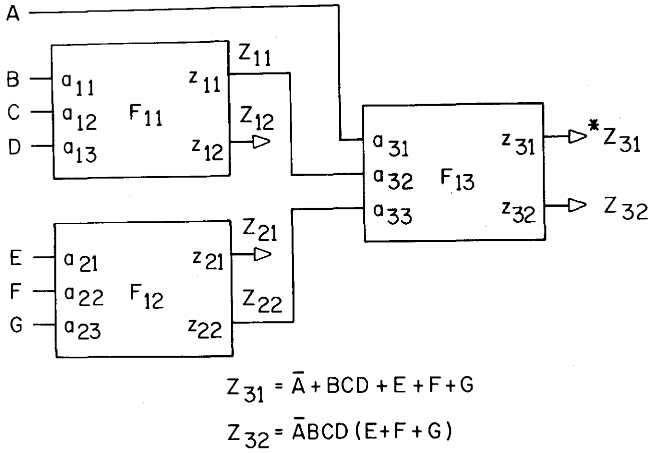
$$Z_1 = \overline{A_1 A_2 A_3}$$
$$Z_2 = \overline{A_1 + A_2 + A_3}$$

Figure 7—A simple LRU for example subsystem

Next we demonstrate the application of converging connections in reducing the number of points which must be observable. Note that in Figure 6 if all AR functions F_i were used in different LRU, all of the n outputs Z_i's must be made observable. In that case, the test pattern T_i however, can be applied simultaneously to each AR function F_i. Thus the number of tests would be equal to the maximum of the number of tests in the test patterns T_i's. On the other hand, by the use of a converging connection we are able to reduce the number of points which must be made observable to one (from n). But the length of the test procedure to obtain the same degree of diagnostic resolution is now equal to (or less than) the sum of the number of tests in all the test patterns T_i $(1 \leq i \leq n)$ and the number of tests in the test pattern T_c for the AR function F_c.

AN ILLUSTRATIVE EXAMPLE

To illustrate the application of AR functions, a very elemental LRU (Figure 7) is selected as a building block of a simple subsystem (Figure 8). Realistic systems, of course, would be considerably more complex. A comparison is made of the AR approach with present day techniques in terms of the required number of



$$Z_{31} = \bar{A} + BCD + E + F + G$$

$$Z_{32} = \bar{A}BCD(E+F+G)$$

Figure 8—An illustrative subsystem
* The points which are to be observable are identified by the arrowhead

observable points and the length of the required test procedure.

A conventional approach

As can be easily inferred from the functions for Z_{31} and Z_{32}, if the observable points are restricted to the final outputs (Z_{31} and Z_{32}), all faults cannot be localized. For instance, the s-a-0 faults at a_{11}, a_{12} and a_{13} and s-a-1 fault at z_{11} are indistinguishable. Similarly, the s-a-1 faults at a_{21}, a_{22}, and a_{23} and s-a-0 fault at z_{22} are indistinguishable. The distinguishability of these faults can be obtained if the outputs Z_{12} and Z_{21} are made observable. Thus to obtain a complete localization we need four observable points.

Table IV shows a conventional test procedure for the subsystem generated by the use of well-known techniques.[4,5,6] The first four tests provide the testing of the faults at the inputs of F_{11}. The conditions at the other inputs are maintained as prescribed to provide a sensitized path to the output Z_{31}. One of these tests also provides an all zero test for F_{12}. The next three tests similarly provide the testing of the faults at the inputs of F_{12}. Note that the input combinations for F_{11}, to provide a sensitized path for the output Z_{22} during these tests, are carefully chosen to simultaneously provide the tests for F_{11} when the output Z_{12} is observable. The eighth test provides partly for the testing of the input a_{31}. Observe that so far only a subset of the total tests required is applied (indirectly) to F_{13}. The conditions corresponding to the remaining tests are applied by the tests 9 through 12. Note the use of carefully chosen input combination at the inputs of F_{12} to simultaneously provide the required test combinations if the output Z_{21} was observable. Finally test 13 is necessary to provide the remaining test combination for F_{12} when output Z_{21} is made observable. Thus we see that a total of thirteen tests and four observable points are required for localization of all faults.

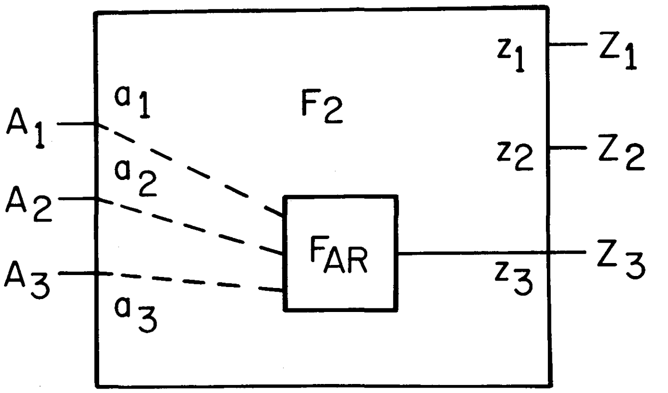
The proposed approach

Next, consider the use of a three-input AR function (F_{AR}) as shown in Figure 9, which is the function discussed in Section 6. One output is to be added to the LRU to make the output of the AR function observable.

Consider the implementation of the subsystem using LRU's with AR functions as shown in Figure 10. Table V shows the required test procedure to locate all faults, except the ones at the outputs Z_{31} and Z_{32}, by observing only the outputs of the three AR functions (Z_{13}, Z_{23}, Z_{33}). Outputs Z_{31} and Z_{32} will be inputs to some other LRU's and therefore their faults will be localized by the AR function of those LRU's. Recall that for each AR function four tests are required (Table III). If the

TABLE IV—A Test Procedure for the Subsystem of Figure 8 (Without AR Functions)

Test No.	Test Procedure							Inputs to F_{13}		
	A	B	C	D	E	F	G	A	Z_{11}	Z_{22}
1	1	0	1	1	0	0	0	1	1	1
2	1	1	0	1	0	0	0	1	1	1
3	1	1	1	0	0	0	0	1	1	1
4	1	1	1	1	0	0	0	1	0	1
5	1	0	0	1	0	0	1	1	1	0
6	1	0	1	0	0	1	0	1	1	0
7	1	1	0	0	1	0	0	1	1	0
8	0	0	0	0	0	0	0	0	1	1
9	0	1	1	1	0	0	0	0	0	1
10	0	1	1	0	0	1	1	0	1	0
11	1	1	1	1	1	0	1	1	0	0
12	0	1	1	1	1	1	0	0	0	0
13	–	–	–	–	1	1	1	–	–	0



$$Z_1 = \overline{A_1 A_2 A_3}$$

$$Z_2 = \overline{A_1 + A_2 + A_3}$$

$$Z_3 = A_1 A_2 + \bar{A}_1 A_3$$

Figure 9—The LRU with AR function

conditions corresponding to this test pattern are applied to each of the AR functions, then the fault localization will be achieved. The first four tests provide the conditions for the test pattern of the AR functions in F_{21} and F_{22}. In so doing, by applying appropriate conditions at the input A we also manage to provide the conditions corresponding to two of the four tests for F_{23}. The last two tests provide the remaining tests for F_{23}. Note that we use the results at the output Z_{33} only if we have determined that no input fault exists at F_{21} and F_{22} i.e., when Z_{13} and Z_{23} do not show a failure. Thus the output Z_{33} is used only to detect and locate the input faults of F_{23}.

TABLE V—A Test Procedure for the Subsystem of Figure 10 with AR Functions

Test No.	Test Procedure							Inputs to F_{23}		
	A	B	C	D	E	F	G	A	Z_{11}	Z_{22}
1	0	0	1	0	0	1	0	0	1	0
2	0	0	0	1	0	0	1	0	1	0
3	1	1	0	1	1	0	1	1	1	0
4	1	1	1	0	1	1	0	1	1	0
5	0	1	1	1	0	0	0	0	0	1
6	1	1	1	1	0	0	0	1	0	1

Thus a test procedure with six tests and three observable points is sufficient to localize the failures. If we would like to reduce the number of observable points further, we can use a converging (Section 7) AR function F_3 such as the 2-input AR function of an earlier section (Figure 11).[3] The points which must be observed are the outputs Z_{33} and Z_{41}. But when we reduce the number of observable points by converging we pay a penalty in the length of test sequence. Recall that, as per upper bound it would take 11 tests to test the convergent structure (as a two input AR function requires three tests). Two additional tests will be required to provide the remaining tests for the AR function F_{23} as before. And thus a total of 13 tests may be necessary. However, as shown in the test procedure of Table VI, the actual number of tests required for the converging connection is only eight, and therefore, a total of nine tests are required (Tests 1 and 5 are identical). Note that the required test pattern for the

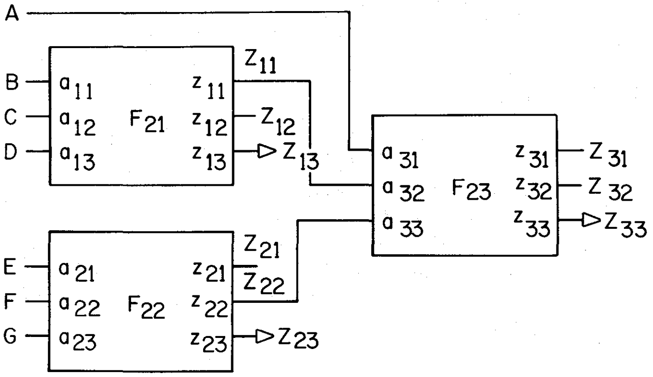


Figure 10—Implementation of the subsystem using LRU's with AR functions

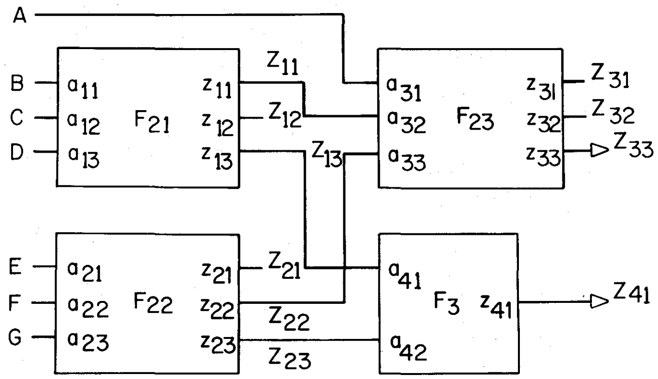


Figure 11—Use of a converging connection to reduce the points which must be made observable

TABLE VI—A Test Procedure for the Configuration of Figure 11

Test	Test Procedure							Inputs to F_{23}			Inputs to F_3	
No.	A	B	C	D	E	F	G	A	Z_{11}	Z_{22}	Z_{13}	Z_{23}
→1	0	0	1	0	0	1	0	0	1	0	0	0
2	0	0	0	1	0	1	0	0	1	0	1	0
3	1	1	0	1	0	1	0	1	1	0	0	0
4	1	1	1	0	0	1	0	1	1	0	1	0
→5	0	0	1	0	0	1	0	0	1	0	0	0
6	0	0	1	0	0	0	1	0	1	0	0	1
7	1	0	1	0	1	0	1	1	1	0	0	0
8	1	0	1	0	1	1	0	1	1	0	0	1
9	0	1	1	1	0	0	0	0	0	1	1	0
10	1	1	1	1	0	0	0	1	0	1	1	0

converging AR function F_3 is applied in the process of applying test patterns for the AR functions of F_{21} and F_{22}. Table VII summarizes these results for the subsystem considered in the illustration.

It is important to note that the AR function design is unrelated to the function actually performed on an LRU, but is only dependent on the number of inputs. Thus in practical LSI applications, as the level of integration goes up and as the gate/pin ratio increases, the percentage redundant logic required for AR function will decrease.

TABLE VII—Comparisons of the Different Approaches to Implement a Subsystem (Figures 8, 10, and 11)

No.	Case	No. of Points Which Must be Made Observable	Length of the Test Sequence for Fault Localization
1	LRU's without AR functions (Figure 8)	4	13
2	LRU's with AR functions (Figure 10)	3	6
3	LRU's with AR functions and a converging AR function (Figure 11)	2	9

CONCLUSIONS

In this paper we have introduced the notion of adding special Ambiguity Resolver functions to Least Replaceable Units in order to provide localization of input/output failures. We have demonstrated the existence of these functions for any number of inputs and shown how they can be synthesized by means of a conformal approach. For those cases in which it is important to further reduce the number of points that must be made observable, a convergent structure is discussed, along with an upper bound on the length of the required test sequence.

The Ambiguity Resolver approach is certainly not the complete answer to the fault localization problem in Large Scale Integrated circuitry, but it is a step in the direction of tailoring the diagnostic technique to the characteristics of the environment.

REFERENCES

1 H Y CHANG E G MANNING G METZE
Fault diagnosis of digital systems
Wiley-Interscience New York 1970 Chapter II

2 G L SCHNABLE R S KEEN JR
Failure mechanisms in large-scale integrated circuits
IEEE Trans on Electron Devices Vol ED-16 No 4 April 1969 pp 322-332

3 J E McCORMICK
On the reliability of microconnections
Electronic Packaging and Production Vol 8 No 6 June 1968 pp 187-189

4 D B ARMSTRONG
On finding a nearly minimal set of fault detection tests for combinational logic nets
IEEE Trans on Electronic Computers EC-15 February 1966 pp 66-73

5 J P ROTH
Diagnois of automata failures—A calculus and a method
IBM Journal of Research and Development 10 1966 pp 278-291

6 W H KAUTZ
Fault testing and diagnosis in combinational digital circuits
IEEE Trans on Computers EC-17 pp 352-366

7 M A MEHTA
A method for optimizing the diagnostic resolution of input-output faults in digital circuits
PhD Thesis Illinois Institute of Technology 1971

Computer-aided drug dosage*

by LEWIS B. SHEINER, BARR ROSENBERG and KENNETH L. MELMON

University of California Medical Center
San Francisco, California

and

University of California
Berkeley, California

INTRODUCTION

Major efforts have been devoted to the application of computational techniques to medical diagnosis, a difficult computational task. The amount of information necessary to perform an exhaustive diagnostic search is formidably large. The "costs" associated with making certain diagnoses or eliminating others often affect a tentative diagnosis as much as the probabilities of the conditions being considered. These "costs" are usually intuitively considered by the physician and usually are not available as numerical quantities to use in a linear programming algorithm. Lastly, diagnosis is done both rapidly and well by most physicians.

In contrast stands therapeutics, specifically the administration of medicines. After deciding what and how much effect is desired (choosing the drug and the therapeutic objective) the problem is how much to give and in what temporal pattern. Drug-dosing is inherently quantitative and is presently done sub-optimally. Accordingly, we have approached the construction of algorithmic techniques for determination of drug dosage.

Since drug effects correlate far better with drug levels in the body than with doses administered, a **T**arget **B**lood **L**evel (TBL) approach to drug administration seems reasonable. In order to implement such an approach, estimates of appropriate pharmacokinetic parameters for individual patients must be obtained so that pharmacokinetic models, providing the mathematical framework for drug level predictions, can be used to determine optimal dosage regimens. We have described a conceptual scheme and associated statistical methodology to accomplish this objective.[1] In this paper, we summarize the scheme and methods, provide justification for our approach, report early results, and describe some future plans.

BACKGROUND

Eighteen to thirty percent of all hospitalized patients sustain one or more adverse drug reactions[2,3] and the duration of their hospital stay is nearly doubled as a result. In addition, 3 to 5 percent of all admissions to hospitals are primarily for a drug reaction[2,4] and 30 percent of these patients have another drug reaction during their hospital stay. The economic consequences of these reactions are staggering: one-seventh of all hospital days is devoted to the care of drug toxicity, at an estimated annual cost of 3 billion dollars.[5] The same amount would cover the cost of all prescription drugs used in this country per year.

An encouraging aspect of the large adverse reaction rate is that 70-80 percent of them are direct extensions of the pharmacologic actions of the drugs involved,[6] and therefore should be predictable and preventable. The majority of the pharmacologic adverse reactions are probably not the result of extraordinary individual sensitivity to the actions of the agent, but rather of dosage regimens leading to inordinately high blood levels.

The evidence which associates drug levels in blood or tissues with their effects is striking. It is particularly clear-cut for those drugs responsible for more

* Supported in part by United States Public Health Service Grants GM 16496 and GM 10791.

than 50 percent of dose-related adverse reactions:[6] digitalis preparations,[7] anti-arrhythmic drugs,[8,9] and antimicrobials.[10] As an example, consider digoxin, a drug which, in clinical practice, is both highly useful and formidably toxic. About 20 percent of all patients receiving the drug demonstrate pharmacologic toxicity and as many as 30 percent of those demonstrating such toxicity may die of it.[11] A three-or-more-fold variation in the blood levels of digoxin is seen in different individuals given the same doses of the agent, but toxicity is highly correlated with blood levels: in one study, 90 percent of individuals with levels less than 2 nanograms/ml of plasma were nontoxic and 87 percent of those with levels greater than this amount were toxic.[12] For digoxin, then, individual sensitivity differences may account for as little as 10 percent of clinical toxicity.

That levels of drugs in body fluids should bear a more constant relationship to effects than doses administered is readily understandable. Drugs act on receptors and elicit effects which are proportional to the fraction of occupied receptors. The concentration of a drug at its receptor is a major determinant of drug effect. The utility of blood levels stems from the observation that although few drugs act on receptors in the blood itself, after drug distribution, tissue concentrations of drugs (presumably at their sites of action) bear a reasonably linear and constant relationship to blood levels. Individual variation in absorption, distribution, metabolism and elimination of drugs, however, causes widely varying blood levels to be found in different individuals, despite standard doses.

It seems clear, then, that a TBL approach to drug dosage design is theoretically sound and likely to be beneficial. Pharmacokinetic models can represent and quantify variations in absorption, distribution, metabolism and elimination of drugs. In order to utilize a pharmacokinetic model for the purpose of drug dosage design, some way of predicting individual variation in the parameters of the model must be available. An obvious approach is to measure blood levels of agents at varying time intervals after administration of a standard regimen and to adjust the parameters of the model to yield a best fit to the observed levels. This is analogous to the way in which most physicians administer drugs today: a standard regimen is begun, effects are observed (presumably proportional to levels) and dosage adjustments are made based upon these observations. The quantitative adjustments are more intuitive than optimal. We can do better than merely improve the quantitative aspects of the feedback process described above, although doing only this can lead to excellent results.[13]

The factors responsible for pharmacokinetic variation can be related to the physiological determinants of pharmacokinetic processes. For example, digoxin is primarily eliminated by renal excretion. The elimination rate of the drug should be, and is, proportional to renal function.[14] Tests designed to assess renal function [such as measuring the serum creatinine or **B**lood **U**rea **N**itrogen (BUN)] are routine. These measurements can be used to modify the dosage regimen of digoxin in accord with the excretory ability of the individual patient.

The combination of prior clinical information with quantitative feedback adjustment should produce a system for optimal drug dosage design. In order to do this we require a black box with the following characteristics: the inputs are clinical data, commonly and easily available (such as height, weight, age, sex, and the results of standard laboratory tests, e.g., tests of renal function); the outputs are a set of predicted values for the parameters of a pharmacokinetic model. On the basis of these predicted parameter values, the initial dosage of a drug can be set to produce the TBL. As additional data in the form of blood level measurements or changes in the laboratory tests become available, estimates of the parameters can be optimally modified (by the black box) to improve dosage suggestions.

THE DOSE—BLOOD LEVEL RESPONSE MODEL

The first task in the design of the black box is to formulate a model for the relationship between the

TABLE I—The Conceptual Scheme

I. Observations (O) → Physiologic Variables (P)
type: usually non-linear
data source: general population
example: Body Surface Area (BSA) = F (height, weight)

II. P → Pharmacokinetic Variables (Q)
type: linear
data source: patients receiving drug
example: Volume of Distribution (Vd) = F (BSA)

III. Q → Pharmacokinetic Parameters (K)
type: usually non-linear
data source: theoretical
example: Rate Constant of Elimination (Kel) = Clearance/Vd

IV. K → Blood Level Predictions
type: non-linear pharmacokinetic model
data source: theoretical
example: one compartment model with first order absorption

clinical data and the pharmacokinetic parameters. Our model is outlined in Table I. Clinical observations (O) are used to predict physiological variables (P). These predictions can be defined through study of the general population and need not be restricted to those individuals who have received the drug of interest. The relationships are not restricted as to type; they may be linear, non-linear, discontinuous, etc. The example given in Table I, **B**ody **S**urface **A**rea (BSA), is a quantity estimated from a non-linear function of height and weight and is expected to be more linearly related to metabolic capacity and body "compartment" sizes than either height or weight. In Table I, the pharmacokinetic model itself is a simple one: the one-compartment open model with first order absorption. Its parameters (K) are: Kel, the first-order rate constant of elimination of the drug; Vd, the volume of distribution of the drug; and a series of Ka's and f's, each pair representing the rate constant of absorption and the fraction of dose absorbed for a specified route of administration. The K may be linearly related to the P. For example, the Vd of a drug might be predictable from a patient's body surface area, as shown in Table I. A non-linear function of a K might bear a linear relationship to one or more of the P. An example is the clearance of a drug (the product of Kel and Vd) which would be linearly related to a P describing renal function (the **G**lomerular **F**iltration **R**ate, GFR) if the drug were eliminated by the kidneys. To keep the scheme general, a set of pharmacokinetic variables, Q (distinct from the K) are defined as functions, again of any variety, of the K. The Q are defined wholly from theoretical considerations. Finally, having defined the P and Q so that we expect linear relationships to hold between them, we assert that they are related linearly. The first data-fitting task is that of estimating the coefficients of the linear P to Q transformation. This can be done utilizing data from individuals whose clinical observations are known and whose blood levels of drug have been measured.

The full model allows utilization of general population data and prior information in the form of both theoretical relationships and constraints on the coefficients to be estimated for the P to Q transformations. The use of an extremely simple pharmacokinetic model is justified on two grounds. First, for clinical medicine, our scheme must be responsive to shifts in patient characteristics and inexpensive to operate. More parameters imply more data-fitting, degrading responsiveness, and increasing cost. Second, most drugs with clinical utility do not cause significant toxicity until their blood levels are at least twice those at which significant efficacy is achieved. Therefore, there is no need for great precision; prediction errors of 50 percent should have little clinical import.

Before discussion of the problem of merging feedback (blood-level) information with initial estimates, some preliminary tests of the appropriateness of the conceptual scheme and of the ability to estimate coefficients for the P to Q relationship will be presented.

TEST OF THE MODEL

Dr. Thomas Smith of the Massachusetts General Hospital kindly supplied us with clinical data, dosage history, measured blood levels (one per patient) and determinations of the presence or absence of toxicity in a group of 99 patients who received digoxin. The clinical data for each patient consisted of age, weight and BUN. Fifteen patients were questionably toxic and 18 definitely toxic by published criteria.[11,12] The model was adapted to digoxin from the available clinical data, by using weight as a predictor of BSA, and age, weight and BUN as predictors of renal function. Vd was assumed proportional to BSA. Digoxin clearance was assumed proportional to GFR and BSA since the non-renal losses of digoxin are presumably metabolic and should therefore be related to BSA. All digoxin had been administered orally so that Ka and f values for the oral route only were needed. Digoxin has an absorption value (f) of .85.[15] While the precise rate of oral absorption is not known, it is sufficiently rapid relative to rates of elimination[14] that an arbitrary large value could be assigned to the oral-rate constant of absorption without expected loss in accuracy. Data was adequate to establish an expected value for Kel for normal individuals, and to support the assumption that digoxin clearance should be linearly related to GFR.[16] Data on the rate of non-renal loss of the drug is scanty, and no clear data exist for estimating the relationship between BSA and Vd. Accordingly, the patient data were first used to estimate these values using standard non-linear data fitting techniques. The blood levels were weighted by the inverse of the observed values plus the measurement error in their determination because, on the one hand, with unweighted measurements, fitting to absolute errors inappropriately magnifies the importance of those patients with large measured blood level values, and, on the other hand, data fitting by pure relative error (corresponding to weighting by the inverse of the square of the measured value plus the measurement error variance) remove all bias toward greater predictive accuracy for higher values, which are the more clinically important ones, being indicative of toxicity.

TABLE II—Relation Between Program-Predicted and Actual Blood Levels of Digoxin in a Sample of 99 Patients

Information Used	Correlation Coefficient (r)	r^2	σr^2
1. Daily dose	.29	.08	—
2. 1 + model	.31	.10	+.02
3. 2 + weight	.42	.18	+.08
4. 3 + age	.47	.23	+.05
5. 4 + BUN	.73	.54	+.31

We arrived at a value of 9 percent per day non-renal loss (as opposed to an estimate of 14.4 percent per day[16]) and a Vd value of 351 liters/square meter of BSA. It is noteworthy that these estimates could be easily obtained from the crude clinical data at our disposal. The use of our scheme permits estimation of population values for pharmacokinetic quantities from usual clinical data and does not demand separate and elaborate experiments. This point is crucial, since an individual blood level determination obtained from clinical usage of the feedback portion of the system will not only serve to improve the estimates of that individual's responses, but also add to the data base, making improved population estimates possible.

With the scheme set for digoxin and necessary values obtained from the literature, or as noted above, from the data itself, the ability of the scheme to predict blood levels of the patients, using various amounts of prior clinical information, was examined. Table II shows the resultant correlations. The only upward bias in utilizing the correlation coefficients to describe the predictive capacity of the system stems from the use of one degree of freedom for the estimation of the non-renal losses, since Vd operates in the pharmacokinetic model as a scale factor only. The fraction of total variance in the actual levels which is explained by the computed levels is expressed by r^2; in the absence of any clinical data save the dose administered, only 8 percent of the level variance is explained, while use of all the clinical data allows explanation of 54 percent of the variance. Knowledge of renal function makes the major contribution to accurate predictions, but the patient's weight, the sole predictor of size used, explains as much of the variation in levels as does the dose. In order to exclude the possibility that the few very high values present in the data were responsible for the high correlation we obtained, despite our weighting scheme, predictions were examined when weighted by the inverse of the square of the measured value plus the measurement error variance. Under these conditions, the correlation coefficient for predictions made with full information fell only to .71.

More important, however, than the ability to predict the measured values with a modest degree of accuracy, is the potential clinical implication of the predictions. As has been mentioned, digoxin levels greater than 2 ng/ml are associated with a significant incidence of toxicity. Table III compares the toxicity predictions of the actual measured levels, using this cut-off point, with those of the predicted levels. Although the predicted values are not quite as good at predicting toxicity as are the measured values (there are, however, no significant differences in the columns of Table III by χ^2 analysis), use of the scheme might have prevented 10 of the 18 toxicities in this group of patients. Clearly the potentially prospective availability of the predictions and the non-toxic dosage schemes they would engender outweigh the slight (retrospective) diagnostic advantage of the actually measured levels.

THE FEEDBACK PROCESS

The requirement for feedback adjustment of the model parameters is that it make an orderly transition from estimates of an individual's pharmacokinetic parameters based solely upon population data to ones based to a greater extent upon measured blood levels of the drug. Our approach is based upon an estimation scheme valid in circumstances where an individual parameter vector characterizing responses (in our case, the Q vector "characterizes" the blood level "response") is dispersed about a population mean (that is, individual vectors are similar, but not identical).[17] This regression method allows efficient estimation of the mean value of the parameter vectors in the population, of the extent of dispersion of ind.v.dual parameters about the population mean, and of the

TABLE III—Comparison of Predictions of Digoxin Toxicity by Actual and Program-Predicted Blood Levels

Levels Used	Predictions of Toxicity False +	 False −	Total Correct
Actual	10/66	3/18	71/84
Program-predicted	11/66	8/18	65/84

values of the parameter vectors characterizing each individual. In our case, population data is used to estimate a mean population Q vector, the appropriate adjustments to be made in this vector for the values of clinical observations, a population variance matrix of parameter prediction error, Λ, and the variance of prediction error associated with individual blood levels, σ^2. An individual's Q vector is assumed to be the sum of the population mean vector adjusted for his set of clinical observations, plus an individual shift vector, originally unknown, with mean value, zero. For initial dosage suggestions, the shift vector is assumed to be zero. For the feedback portion of the system, given Λ and σ^2 and the actual versus predicted blood level for the individual, a non-zero value is assigned to the individual shift vector so as to maximize the prediction likelihood function (an empirical Bayes estimator). A more complete description of the statistical methods underlying this approach can be found in References 1 and 18. What occurs, in effect, is that the information content of a blood level measurement is used to decrease the uncertainty of our knowledge of the components of an individual's Q vector (and hence his K vector) so as to minimize the mean square error in the resulting blood level predictions.

TEST OF THE ESTIMATION SCHEME

Examination of the problems attendant upon estimation of a population dispersion matrix has begun. The simpler case of an underlying response model linear in the explanatory variables (the P) was studied first.

Despite the linearity of the underlying model, an analytic solution is not apparent for the dispersion matrix; thus, even in the linear case, one is forced to use non-linear techniques to maximize the likelihood function. For this reason the linear and non-linear cases share many characteristics. If efficient estimation of a dispersion matrix could not be accomplished in the linear case, we would be pessimistic about the ability to do so in the non-linear case. The problem can be reduced to a search over the space of triangular matrices, T, such that $TT' = \Lambda$. The number of parameters to be estimated by non-linear methods is therefore $p = k(k+1)/2$, where k is the dimension of Λ. The Fletcher-Powell algorithm has been used to seek a maximum of the likelihood function over this space. The results of similations to date have been uniformly encouraging. The likelihood function is smooth and well behaved, and is well described by analytical derivatives. Regardless of the initial estimate chosen for T, the search algorithm converges to the same maximum and the same estimate for Ω, indicating that the likelihood function has a unique extremum. Convergence appears to require fewer than 3p iterations in all but a few cases, and 4p iterations has been the maximum. Thus the non-linearity of the problem appears to pose no difficulties.

More important, a relatively small amount of data appears sufficient to generate a satisfactory estimate of Λ. With four simulated observations on each of forty individuals, the estimates have consistently approximated the relative magnitudes of the diagonal elements of Λ, and the signs and magnitudes of the off-diagonal elements.

CURRENT WORK

The preliminary results obtained in the development of both the conceptual and estimation schemes has been encouraging. For the initial prediction portion, it was a simple matter to write an interactive program to run in a time-shared mode (for use via a remote access terminal) which would accept clinical observations on a patient as well as the (physician determined) TBL and deliver a dosage scheme (at present, for digoxin only) adjusted for the individual, and designed to produce the TBL. We note that although it is possible to provide information on the usual range of therapeutic blood levels for a drug through the program, the actual TBL for an individual patient should not be pre-programmed. It is precisely in the choice of this value that the physician is called upon to exercise his judgment as to the severity of the condition being treated, the therapeutic objectives sought and the projected sensitivity of his patient.

The interactive program and the initial dosage suggestions are being tested in a prospective fashion. Out-patients in need of digitalis or in need of adjustment of digitalis therapy are being studied. Both the primary physician and the program are suggesting regimens and predicting blood levels for these patients. The program output is made available to the physician for a randomly selected portion of the patients, and the physicians administer the drug as they see fit. Comparisons will be made in the accuracy of blood level predictions by the physicians and the program, and in the results of treatment (both for efficacy and toxicity) obtained by administration of program-suggested regimens.

The estimation scheme is being further tested by Monte Carlo studies. Multiple blood level data is being obtained on patients receiving digoxin so that the procedures for estimation of Λ can be tested. The questions of major interest are: (1) How much data is necessary to estimate Λ and how valuable is knowledge of the population dispersion in predicting individual blood levels? (2) What is the "trade-off" between clinical observations and subsequent blood level determinations; e.g., how many blood level determinations does it take to make up for lack of knowledge of renal function? (3) Do blood levels drawn early in the course of therapy, when maximum blood levels are not yet attained and toxicity is highly improbable, contribute substantially to the ability to predict future (maximum) blood levels? (4) At what point is the further determination of blood levels unnecessary; that is, when do they cease to contribute new information? We expect to have partial answers to these questions for digoxin at the time of the Spring Joint Computer Conference, and will devote the major portion of our presentation to discussing them.

SUMMARY AND CONCLUSION

We have devised a conceptual scheme which successively relates routinely available clinical observations to measures of underlying physiologic and pharmacokinetic factors and, ultimately, to pharmacokinetic parameters. These parameters can be used in a pharmacokinetic model to produce optimal initial dosage suggestions tailored to individual needs. We have also proposed a statistical methodology for adjustment and improvement of individual estimates in the light of subsequent response data. Preliminary tests of the various portions of the system have been uniformly encouraging. The scheme is highly general; most of the definable influences on drug absorption, distribution, metabolism and elimination can be represented, quantified, merged with others and tested in relation to blood level data. As a consequence of the statistical techniques used and the degree to which extensive exploitation of prior information is possible, determination of even a single blood level for an individual should markedly improve the system's predictive accuracy for him. Drug levels determined in order to improve treatment for an individual also contribute to the underlying data base used to update the various estimated population coefficients. Thus, the system can learn. A broad definition of what constitutes clinical observation will allow us to use the pharmacokinetic variables determined for one drug, for one patient, in the estimation of the pharmacokinetic parameters for another drug for him, if a meaningful relationship holds between them. Hence, the system is a research tool.

REFERENCES

1 L B SHEINER B ROSENBERG K L MELMON
Pharmacokinetic modelling for individual patients: A practical basis for computer-aided therapeusis
Comp and Biomed Res submitted 1971

2 L G SEIDL G F THORNTON J W SMITH et al
Studies on the epidemiology of adverse drug reactions III. Reactions in patients on a general medical service
Bull Johns Hopkins Hosp *119:* pp 299-315 1966

3 B C HODDINOTT C W GOWDEY
W K COULTER et al
Drug reactions and errors in administration on a medical ward
Can Med Assoc J *97:* pp 1001-1006 1967

4 N HURWITZ
Admissions to hospital due to drugs
Br Med J *1:* pp 539-540 1969

5 K L MELMON
Preventable drug reactions—causes and cures
New Eng J Med *284:* pp 1361-68 1971

6 R I OGILVIE J RUEDY
Adverse drug reactions during hospitalization
Can Med Assoc J *97:* pp 1450-1457 1967

7 T W SMITH V P BUTLER E HABER
Determination of therapeutic and toxic digoxin concentrations by radioimmunoassay
New Eng J Med *281:* pp 1212-1216 1969

8 J T BIGGER D H SCHMIDT H KUTT
Relationship between the plasma level of diphenylhydantoin sodium and its cardiac antiarrhythmic effects
Circ *38:* pp 363-374 1968

9 J KOCH-WESER S W KLEIN
Procainamide dosage schedules, plasma concentrations, and clinical effects
JAMA *215:* pp 1454-1460 1971

10 C M KUNIN
A guide to use of antibiotics in patients with renal disease—A table of recommended doses and factors governing serum levels
Ann Int Med *67:* pp 151-158 1967

11 G A BELLER T W SMITH
W H ABELMANN et al
Digitalis intoxication—A prospective clinical study with serum level correlations
New Eng J Med *284:* pp 989-997 1971

12 T W SMITH E HABER
Digoxin intoxication: The relationship of clinical presentation to serum digoxin concentration
J Clin Invest *49:* pp 2377-2386 1970

13 L B SHEINER
Computer-aided long-term anticoagulation therapy
Comp and Biomed Res *2:* pp 507-518 1969

14 J E DOHERTY W H PERKINS
M C WILSON et al
Studies with tritiated digoxin in renal failure
Am J Med *37:* pp 536-544 1964

15 J E DOHERTY
The clinical pharmacology of digitalis glycosides: A review
Am J Med Sci *255:* pp 382-414 1968

16 R W JELLIFFE
A mathematical analysis of digitalis kinetics in patients with normal and reduced renal function
Math Biosciences *1:* pp 305-325 1967

17 C R RAO
The theory of least squares when the parameters are stochastic
Biometrika *52:* pp 447-458 1965

18 B ROSENBERG
Linear regression with stochastically dispersed parameters
Biometrika submitted 1971

Automated therapy for nonspeaking autistic children*

by DAVID CANFIELD SMITH, MALCOLM C. NEWEY and KENNETH MARK COLBY

Stanford University
Stanford, California

CHARACTERISTICS OF AUTISTIC CHILDREN

Earlier publications described our computer method for stimulating language development in nonspeaking children, sketched several case histories (Colby 1968[4]), and gave statistical evidence that our high rate of success (71 percent) was due to our treatment method (Colby and Smith 1970[6]). This paper presents some proposals for making the method more widely available.

The term "Childhood autism" refers to a psychiatric category of mental disorders occurring in children. Though the category is somewhat unreliable, there is a set of characteristics generally associated with autism. Some of the characteristics of autistic children that influenced our approach and our choice of them as subjects are described briefly. More complete descriptions are provided by (Rimland 1964[4]), (Wing 1966[14]), (Ornitz and Ritvo 1968[10]), (Colby 1968[4], Colby and Smith 1970[6]).

An autistic infant's early disinterest in play or being picked up develops into a clear unresponsiveness to people during his second and third years. He avoids contact with other people, sometimes running and hiding from strangers. He does not respond when spoken to, and may be diagnosed as deaf. He may be fascinated by spinning objects and machines, such as phonographs, washing machines, light switches, etc. Far from being mentally retarded, an autistic child may be more adept than normal children at music and mathematics, and may have a better memory.

But the most characteristic attribute is his inability to acquire language. An autistic child has great difficulty understanding the relation between human vocal sounds and meaning. Those that do develop some speech have trouble advancing beyond nouns designating concrete objects to pronouns and verbs with their inherent abstractions. All studies agree that the prognosis for an autistic child is closely correlated with speech development; an autistic child who has not developed speech by the age of five has in the past almost invariably been institutionalized for the rest of his life. If a child has speech, therapy for his other disorders has a far greater chance of success; indeed, absence of language may in itself account for many of his other shortcomings. Having speech, he may begin to play with other children, to go to school, and to build a model of the world that is no longer devoid of concepts transmitted through language.

Though autism is admittedly an imprecise category, the best estimates currently available indicate the incidence of autism to be 4-5 per 10,000 children, about the same incidence as deafness and blindness in children.

The main thrust of our effort, then, is to stimulate language development using a method which is acceptable to an autistic child on his own terms. We give him a chance to play with a machine which happens to be a linguistic entity.

A COMPUTER METHOD FOR TREATING AUTISTIC CHILDREN

While our eventual goal is to implement our method on a system with modest or no computer support, our initial approach was dictated by the resources available to us and by our need for flexibility and modifiability. For these reasons we used the large computer system at the Stanford Artificial Intelligence Project, Stanford University. That system is a time-shared

* This research is supported by Grant PHS MH 06645-10 from the National Institute of Mental Health, by (in part) Research Scientist Award (No. 1-K05-K14, 433) from the National Institute of Mental Health to the third author and by (in part) the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183).

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

DEC PDP-6/PDP-10 combination (DEC 1968[8]). The PDP-6 is primarily used as an I/O driver at Stanford; it controls digital-analogue and analogue-digital converters, among other input/output devices. The faster PDP-10 serves as the central processor. Actually a time-sharing system is not particularly suited to our method, because the need for rapid responses of long duration imposes real-time constraints which are incompatible with the philosophy of time-sharing. In fact, our need to output continuous sound of up to three seconds duration has not yet been implemented to our complete satisfaction.

Our goals of a flexible, easily-modifiable system with a minimum of programming effort initially led us to choose a high-level programming language MLISP (Smith 1970[12]), a dialect of LISP. Admittedly this was like using a sledge hammer to kill a fly, but MLISP enabled us to concentrate most of our early effort on the program itself and not on developing a programming system. Subsequently, our entire system was rewritten (by Malcolm C. Newey) in machine language. It was decided to invest the additional programming effort to gain the greater efficiency of machine language because the developmental phase of many of the games had been completed, and they are no longer being modified. In addition to the basic program itself, we and others at Stanford have developed a set of programs and program modules that facilitate the construction and modification of games. Among these utility programs are a drawing program, with which figures and animation can be easily drawn on a display using a light pen, and a sound recording program with which sound of up to three seconds duration may be recorded, digitized, and stored on the disk.

For hardware we use a computer-controlled Information International Inc. (III) graphics display with a screen about 2′×2′ in size that is refreshed 30 times per second. The need for rapid response, including animation, argued against the economy of a storage-tube display. At present, input is by an alphanumeric keyboard resembling a typewriter keyboard; the child presses a key, and a response is generated. We are currently investigating the possibility of light-pen interaction, so that the child can draw his own figures, and of microphone interaction, so that the computer would be activated by a voice signal rather than a keyboard signal. An associated, rapid-response auditory display is generated by randomly accessed, digitized records of voice, animal and mechanical sounds. The sounds are recorded using one of the utility programs, digitized at 20KC, and stored on secondary storage (IBM 2314 disk). About 1000 audio-visual experiences are available in the currently used set of games, requiring nearly 5 million words of disk storage. When a sound is to be played, it is retrieved from the disk, passed through a digital-to-analogue converter, and output through a speaker system. The high recording rate (20KC) was arbitrary; we wanted good clear fidelity, but a lower frequency (10KC or 5KC) could be used since we tend to favor low, vibrant sounds.

Thus we utilize at Stanford a PDP-10, a III display with keyboard, an IBM 2314 disk drive with 5 million words of secondary storage, and a D/A converter with a speaker system. Clearly these heavy requirements make the widespread replication of our total system impossible. However, we have anticipated eventual implementations of the method on small machines, and even on non-computer-controlled devices (as discussed below), by a modular construction of our programs. The sound and display files are organized into separate and distinct "games". Each game emphasizes a particular aspect of linguistic development. In most cases they do not share files; each game may be separately loaded into the computer much the same (conceptually) as tape cassettes may be loaded into a tape recorder. (In fact, that is one possible implementation of a non-computer device.)

The different game modules may be summoned by the sitter, a non-professional adult who is present at all therapy sessions. The following descriptions of the games are from an earlier paper (Colby and Smith 1970[6]). The descriptions are still valid because, as was mentioned earlier, these games are in their final form. Many games continue to be developed, but their philosophy is similar to that of the games presented here. These games are better described by the slides and recordings which will accompany presentation of this paper.

Game 4 is especially worth noting in view of the autistic child's impaired affective relationships with other people. Elements of compassion are introduced obliquely in several of the frames. For example, in one series a cartoon of a large ice cream cone is accompanied by sounds of loud, happy slurping, and the word SLURP! Then the top scoop falls off (PLOP!), and there is a groan of dismay. The sad faces of many children viewing this reveal clear communication of a shared emotional experience.

Game 1: Letters and numbers. The letter, number or symbol of the pressed key is displayed accompanied by a male or female voice pronouncing the appropriate sound. Many of the letters are shown in different sizes and shapes, and a few are drawn in motion automatically on the video screen.

Game 2: Words. Single words appear on the screen in response to pressing a key. The words consist of verbs and adjectives involving affect and motion, e.g., 'laugh', 'jump', and 'angry.' The affect terms are not only stated but enacted, i.e., the voice laughs while saying 'laugh.' All the major positive and negative affects are referred to in an attempt to exercise the child's affective repertoire. For example, the word 'kill' is spoken very belligerently, and the combination 'XYZ' is accompanied by a contemptuous Bronx cheer.

Game 3: Phrases. A letter or number is displayed in response to the corresponding key being pushed. Following the symbol appears a word or a phrase containing it one or more times, with arrows pointing to these occurrences. A male or female voice states the word or phrase. The words and phrases are selected on the basis of common occurrences in the child's life (e.g., 'funny', 'upstairs') and because of their absurdity (e.g., 'fried filet from Farnsworth', 'lambs go baaaaaa!').

Game 4: Cartoon pictures. This game consists of displayed phrases and pictures along with human, animal, machine and other sounds such as music. The figures consist mainly of animals and machines, with very few humans. Crude motion is achieved by moving the animals' legs or moving a vehicle across the screen. The pictures and sounds are intended to be ludicrous, absurd and amusing to children.

Game 5: Phrase completion. A phrase or sentence consisting of several words appears on the screen in response to a key; e.g., pressing 'J' produces 'the cow jumps over the moon.' A male voice expresses the sentence over and over until the child stops it by pressing any other key. When 'J' is pressed again the same thing happens, but on the third occurrence, while the full sentence appears, the voice omits part of it, saying e.g., 'the cow jumps over the ———.' The intent is for the child to fill in the omitted word or phrase, which is omitted one of three times.

Game 6: Word construction. Here the letters of the alphabet appear in a square around the edge of the video display. On pressing a key, a trumpet sounds with the Los Angeles Rams 'Charge!' call, a female voice says, 'Here comes the word,' a male voice states the word, e.g., 'cat,' and another male voice pronounces the letters, which one by one light up and move to the center of the screen to form the word. The word is again stated when the spelling is completed. The game demonstrates how words are put together letter by letter.

Game 7: Phrase construction. The alphabet is displayed at the bottom of the screen. In response to pressing a key, letters move to the top of the screen, following individual paths, to form a phrase, one word at a time. The effect is of chaotic motion resolving itself at the end into a coherent word.

Game 8: Typewriter. The video portion of this game consists of the symbols of the keys pressed appearing in lines from left to right until the screen fills up, whence they are all erased and a new sequence begun. The audio portion involves a male voice offering comments, requests and questions regarding the displayed symbols—e.g., 'Is this really eight?', 'where is O?', 'see the four?', 'now find I.'

Game 9: Spelling. The child types freely. If, by chance or design, he spells any of the words which occur in any of the games, the rest of the screen is erased temporarily leaving the word he has spelled centered in the screen. The appropriate sound is then played.

We are continuing to experiment with new games in an effort to build a library of the most successful approaches to stimulating language.

THE RATIONALE FOR AUTOMATED THERAPY

Colby has already expressed our initial motivation for using a computer in automated therapy (Colby 1968, p. 642[4]):

> My group's interest in a computer-based method for developing language in nonspeaking disturbed children derived from several sources. First, we were interested in the general problem of using computers in the problems of psychiatry, as for example through computer simulation of belief systems (Colby 1967a[2]) (additional references: (Colby and Smith 1969[5]), (Colby et al. 1971[7])) and man-machine dialogues (Colby and Enea 1967b[3]). Second, the work of Suppes (Suppes 1966[13]) and Moore (Moore 1963[9]) indicates that normal children learn reading, writing, set theory, and arithmetic rapidly and enjoyably by means of computer-controlled keyboards and displays. Third, the observation of many workers regarding the great preoccupation of some disturbed children with mechanical objects which they can manipulate and control is impressive. Since language acquisition in a normal child results from interactions with people (relations which disturbed children find difficult), perhaps nonspeaking disturbed children would find a machine such as a computer-controlled keyboard and display a more acceptable source for linguistic interaction. Hence, an effort was made to take advantage of a child's fascination with machines by providing him

> with a speaking and writing machine to play with. Instead of a person controlling a child, the child can control the machine, making it talk and display symbols at his will.
>
> Language is often described as used for expression and as an instrument for social influence. But during normal language acquisition, it is also used by children as a toy. The method used in the present studies offered each child a means of playing with language. The hunch that children might enjoy this activity was further supported by some preliminary experience with normal children who delighted in the play and whose speech was greatly excited by it during and after the sessions. If a nonspeaking disturbed child could become interested in this sort of play and begin to enjoy developing language as play rather than work, the hope was that he would transfer his use of language from a computer context to other social contexts. If a disturbed child talks, a greater chance exists for understanding what troubles him and for helping him.

In addition to eliminating the fear of interpersonal relationships possessed by many autistic children, the computer has other intrinsic advantages over a human therapist. Some children's linguistic difficulties stem from an inability to abstract from complex and changing situations. They are baffled by vocal sounds coming from people. Even the most well-intentioned human therapist will inject variety into the learning situation, thereby including one more variable with which the child must deal. The response of the machine is an absolute constant which forms a firm base for the child to build upon. We frequently observe children striking the same key over and over, and listening intently to the sound. "Adults do not tolerate repetition well, and when a child strikes the same key for twenty minutes, it takes great control not to interfere. We know from experience that if a sitter tries to stop repetitions, the child will resist the interference and refuse to play further. And he is right." (Colby and Smith 1970[6])

> Has a nonspeaking autistic child lost hope of understanding and using language? Has he given up so that nothing seems worth the effort or risk? If so, we get him to try again, to rekindle the hope by providing him a linguistic experience requiring little effort and at which he cannot fail. For a small amount of effort in pressing a key, he produces a large effect; and there is no risk of failure because something always happens, and he is never corrected for being wrong about what happens. He is free to select those symbols he wishes to manipulate. He is free to exercise his curiosity, to explore, to try to make sense out of the games. The experience is designed to be fun, non-serious, not like school (BLEAGGH!). The program is full of foolishness, nonsense and absurdities intended to delight a child and to elicit glee and exuberance. Only personal observation of the children playing with the display can convince adults of how pleasurable it can be. (Colby and Smith 1970[6])

The strongest argument, though, for our method is the favorable results we have achieved. Other treatment methods for nonspeaking autistic children report an improvement rate of 3 percent-57 percent (Bettelheim 1967[1]). To date (Nov. 1971) we have treated 21 cases over the past four years. We have estimated 15 (71 percent) to have improved in language development. We classify a child as nonspeaking if he is mute or utters only a word or two per month or year. He will also generally utter other sounds that are unintelligible. (However, all but one of our children have shown some language comprehension.) We rate a child improved if he moves from this state to volunteering appropriate intelligible words and phrases in social communication. The speech must be non-echolalic and be initiated by the child with the intention of communicating information. Though the enunciation need not be perfect, it must be sufficiently good so that the child can make practical use of language in interpersonal relations. Sometimes the children offer quite well-constructed sentences, but more often they utter short, primitive sentences or phrases.

AN ECONOMICAL THERAPY MACHINE (ETM)

Our implementation of an automated therapy laboratory on the PDP-10 is clearly not feasible for most clinics, schools or even hospitals around the country. The computer requirements described above would cost well over a million dollars if duplicated faithfully. But this approach is not even desirable. Our goal was to use whatever resources were available to experiment with automated therapy techniques, bypassing as many developmental problems as possible. Now that we have had several years of experience with this type of treatment, we can begin to specify the characteristics of an economical therapy machine (ETM). By this we mean a machine suitable for use in a large number and variety of institutions dealing with language problems.

First and foremost is effectiveness; the machine must be at least as effective as our system in treating language disorders. Thus it must have many of our system's features which we have found to be important: (1) a video screen and provisions for showing still pictures and (at least crude) motion. (2) An audio device capable of producing continuous sound in synchronization with the pictures. (3) A keyboard or similar device with which the child may select a picture/sound, plus the appropriate logic for selecting his choice from the storage. (4) Storage devices for pictures and sounds; they could use the same device (e.g., movie film with sound track) or separate ones. The storage devices and their accessing logic must have at most a one second delay (⅓ second is optimal) between the time the child requests a picture/sound and the time it is played for him. The child's attention span will not tolerate a longer delay.

Next in importance is low cost. The machine must be economically feasible for a wide range of institutions; individual machines can and should cost no more than \$1000-\$5000. It may be possible to group several terminals together under one system, in which case a small computer (e.g., a PDP-8) could be used for the logic. A system with five terminals should cost around \$15,000. These are only rough estimates; they might be improved upon.

Other essential characteristics are that the machine be reliable, rugged (the keyboard, at least, must be able to withstand rough handling by children), easily and cheaply repairable (hopefully using stock, readily-available parts), portable, self-contained and requiring no special environmental controls (such as air conditioning or humidity control).

This is admittedly a formidable list of requirements, but one which we believe to be well within the capabilities of current technology. Some suggested implementations are presented below.

There are other characteristics that are desirable but not essential in a therapy machine: (1) Color in the pictures. (2) Better motion than the 2-4 frame motion we use; also more professional cartoon characters. (3) Longer sound than our current three-second time limit. (4) Flashing lights and changing geometric patterns, accompanied by music; the children are fascinated by orderly motion. (5) More physical involvement while participating in the games. Many children are hyperactive and squirm and fidgit when required to sit in a chair to play the games. If they could run around the room punching buttons or jumping on pedals to activate the machine, it would not only be more enjoyable to these children but would probably increase their attention span as well. (6) Finally, custom tailoring is important. It should be easy to construct a new game specifically for an individual child. Frequently a child will show interest in a particular sound or word or phrase; it should be possible to construct a new game exploiting this sound for the child's next therapy session.

All of these features are designed to hold the child's interest and to stimulate his involvement with the machine. Variety and novelty are the key characteristics. Any suggestions the machine designer would have would, of course, be welcome.

POSSIBLE IMPLEMENTATIONS OF AN ETM

The following are some rough ideas we have had on the form an economical therapy machine might take.

The video screen could be most simply realized as a standard slide-projector screen. This would exclude some versatility, such as permitting the child to draw on a display screen with a light pen, but the cost would be very low.

The pictures could be generated using a standard film strip driven by a high-speed stepping motor. The motor would have to be fast enough to advance over several frames of the film to get to the desired frame within the ⅓-1 second time constraints. It should also be able to stop, so that a single frame remains shown, and to proceed at a slow rate suitable for showing motion. Alternatively, a plate of fiche with the pictures imprinted on it would be used together with a scanning head. Both of these approaches require the design of logic circuitry to control and count the stepping of the motor or the positioning of the scanning head. This circuitry constitutes the main non-standard aspect of the ETM. It has to be designed from scratch; but presumably, once the design costs have been met, it could be constructed inexpensively using low-cost integrated circuit chips.

The sound could perhaps most easily be recorded on a sound track on the film strip. This requires that the film strip be constantly moving, even for still pictures. The advantage is that each game could be packaged in a self-contained and easily-changeable cassette. The problem of constructing new games tailored to individual children reduces to the problem of making new cassettes; there is much inexpensive movie-with-sound recording equipment available today. Alternatively, a separate sound tape could be used in conjunction with the picture producing device. This would complicate the controlling circuitry, since two tapes would have to be coordinated (one for the pictures, one for the sound). A small computer might become feasible with this approach, particularly if several terminals were to be used. Another approach is to have a rapidly rotating drum on which the sound, and possibly the pictures, is recorded. Movable or fixed read/write heads would be used to access the sound; a high RPM would insure that latency is sufficiently small. This arrangement is particularly suited to time-sharing several terminals using a small computer; one copy of the sounds could be used by all the terminals connected to the drum.

We envision a very wide market for a machine designed along these lines. Many hospitals, clinics and schools are becoming increasingly concerned with correcting language disorders. Furthermore, normal children thoroughly enjoy our system. If used in the schools, it could provide a powerful stimulus for their early interest in language, leading perhaps to increased reading efficiency at an early age.

REFERENCES

1 B BETTELHEIM
The empty fortress
The Free Press New York 1967

2 K M COLBY
Computer simulation of change in personal belief systems
Behavioral Science 12 248-253 1967h

3 K M COLBY H ENEA
Heuristic methods for computer understanding of natural language in context-restricted on-line dialogues
Mathematical Biosciences 1 1-25 1967b

4 K M COLBY
Computer-aided language development in nonspeaking children
Archives of General Psychiatry 19 pp 641-651 1968

5 K M COLBY D C SMITH
Dialogues between humans and an artificial belief system
Proceedings of the International Joint Conference on Artificial Intelligence The MITRE Corporation Bedford Massachusetts 1969

6 K M COLBY D C SMITH
Computer as catalyst in the treatment of nonspeaking autistic children
Computer 4 47-53 1971

7 K M COLBY S WEBER F D HILF
Artificial paranoia
Artificial Intelligence 2 1-75 1971

8 *PDP-10 system reference manual*
Digital Equipment Corporation Maynard Mass 1968

9 O K MOORE
Autotelic responsive environments and exceptional children
Responsive Environments Foundation 1963

10 E M ORNITZ E R RITVO
Perceptual inconstancy in early infantile autism
Archives of General Psychiatry 18 pp 76-98 1968

11 B RIMLAND
Infantile autism
Appleton-Century-Crofts New York 1964

12 D C SMITH
MLISP
Stanford Artificial Intelligence Project Memo AIM-135 Stanford University 1970

13 P SUPPES
The uses of computers in education
Scientific American 215 pp 206-220 1966

14 J K WING
Early childhood autism
Pergamon Press New York 1966

An inferential processor for interacting with biomedical data using restricted natural language*

by DALE W. ISNER

University of Pittsburgh
Pittsburgh, Pennsylvania

INTRODUCTION

This paper describes the design and implementation of a natural language system (hereafter referred to as ENGOLISH) which was developed to support and enrich the utilization of computer models of physiological systems. In addition to providing capabilities for interrogating data bases using a subset of English, ENGOLISH also provides capabilities for defining and structuring data bases using this subset of English. In particular, this facility has been used to encode relevant information from documents concerning these models to enrich their utility. ENGOLISH is structured around GOL (**G**oal **O**riented **L**anguage) which is a programming language developed by H. Pople to provide a means in which to express algorithms involving heuristic search and problem solving. GOL provides not only the basis for organizing and searching ENGOLISH data structures but also provides the basis for ENGOLISH's heuristic parsing algorithms. For a description of GOL, see Pople.[13]

The development of retrieval systems, using subsets of English to interrogate structured files of data, has received considerable attention. Simmons[23] summarizes much of this work. In general those retrieval systems referred to as "fact-retrieval systems" or "question-answering systems" consist of the following basic components:

(1) A query language (in this context a subset of natural language).
(2) An internal language or data structure in which facts may be represented or encoded.
(3) A translator for translating questions posed in the query language into programs or subroutine calls to search the internal data structures.

In addition to the above basic components, the following are regarded as features which are desirable in such systems:

(1) A general internal representation (ability to add new and perhaps differently structured data to the system without major modifications).
(2) Ability to build and expand the data base using the query language.
(3) Ability to alter or extend the query language.
(4) Inferential capability, including the ability to handle quantifiers.
(5) Capability for dealing with modalities and state spaces.

In regards to such systems, Schwarcz et al.[17] have identified the need for solving the following three major interrelated problems.

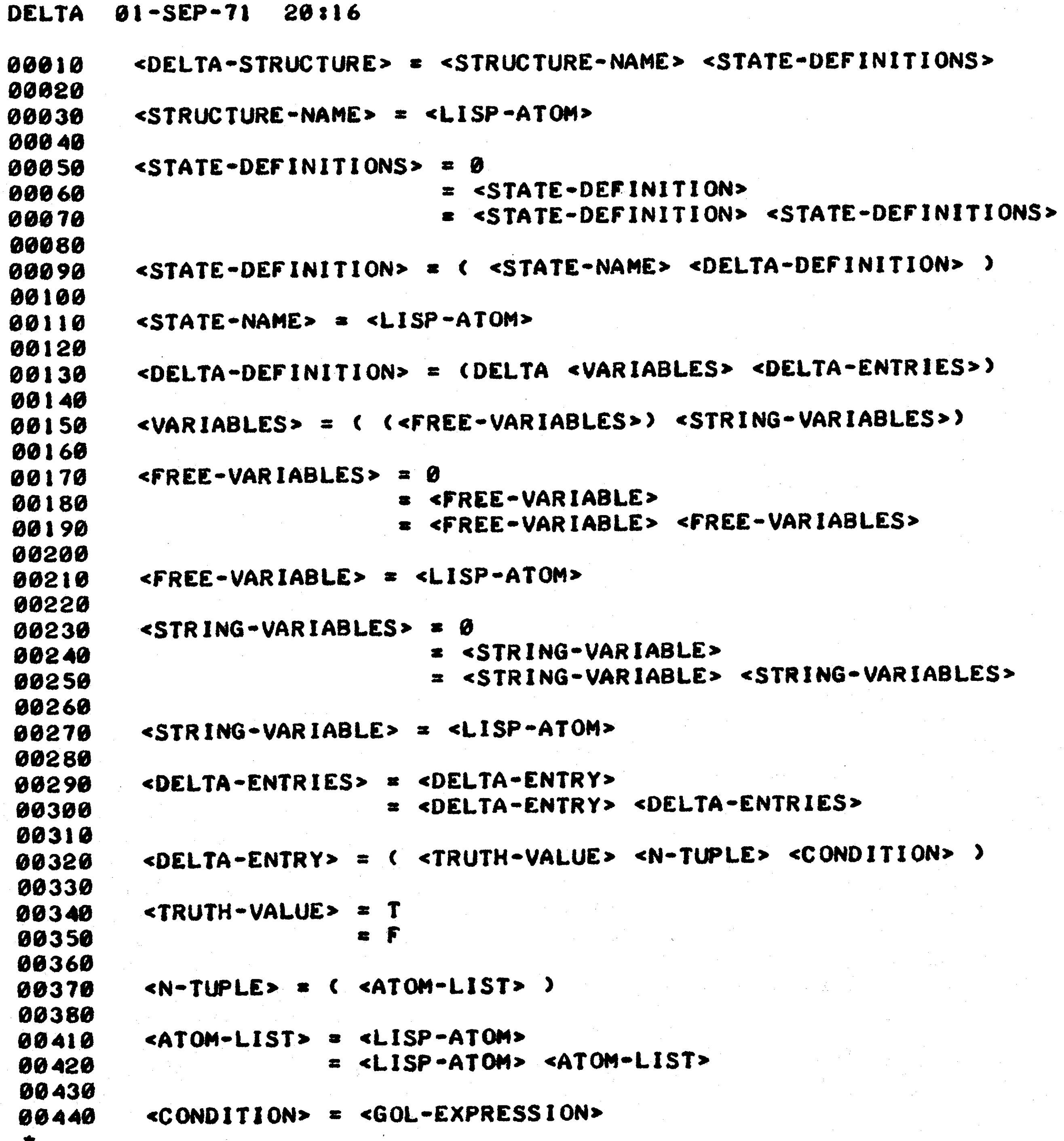
1. The development of a formalized data representation language in which the aspects of meaning that are relevant to the system's uses are explicitly and unambiguously represented;
2. The development of algorithms for translating inputs in the English subset into appropriate storage and retrieval commands for operation on the data base, making appropriate use of contextual information to resolve both syntactic and semantic ambiguity, and for translating structures retrieved from the data base into well-formed English answers;
3. The development of algorithmic and/or heuristic methods for determining what subset of the data base is relevant to filling a given retrieval request and, using the logical relationships

* The work incorporated in this report was carried out under NIH Contract 70-4121 and also supported in part by NIH grant PM 00049-02.

represented in the data base, for deducing answers from this subset to meet the request.

SEMANTIC ORGANIZATION AND EXAMPLES

The primary consideration in the development of a question-answering system is the definition of the data structures which are to be used for its semantic basis. Currently most such query systems use one of two approaches. The first (hereafter referred to as a "logic-based system") uses a logic as its basis, usually a first order theory, and employs a theorem prover as its means for searching and inferring facts from the data base. (For examples see References 6 and 16.) The



```
DELTA   01-SEP-71   20:16

<DELTA-STRUCTURE> = <STRUCTURE-NAME> <STATE-DEFINITIONS>

<STRUCTURE-NAME> = <LISP-ATOM>

<STATE-DEFINITIONS> = 0
                    = <STATE-DEFINITION>
                    = <STATE-DEFINITION> <STATE-DEFINITIONS>

<STATE-DEFINITION> = ( <STATE-NAME> <DELTA-DEFINITION> )

<STATE-NAME> = <LISP-ATOM>

<DELTA-DEFINITION> = (DELTA <VARIABLES> <DELTA-ENTRIES>)

<VARIABLES> = ( (<FREE-VARIABLES>) <STRING-VARIABLES>)

<FREE-VARIABLES> = 0
                 = <FREE-VARIABLE>
                 = <FREE-VARIABLE> <FREE-VARIABLES>

<FREE-VARIABLE> = <LISP-ATOM>

<STRING-VARIABLES> = 0
                   = <STRING-VARIABLE>
                   = <STRING-VARIABLE> <STRING-VARIABLES>

<STRING-VARIABLE> = <LISP-ATOM>

<DELTA-ENTRIES> = <DELTA-ENTRY>
                = <DELTA-ENTRY> <DELTA-ENTRIES>

<DELTA-ENTRY> = ( <TRUTH-VALUE> <N-TUPLE> <CONDITION> )

<TRUTH-VALUE> = T
              = F

<N-TUPLE> = ( <ATOM-LIST> )

<ATOM-LIST> = <LISP-ATOM>
            = <LISP-ATOM> <ATOM-LIST>

<CONDITION> = <GOL-EXPRESSION>
*
```

Figure 1

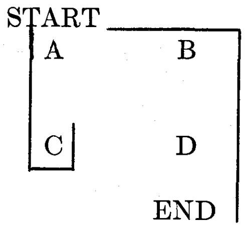
second approach utilizes networks for representing the primary relations concerning its data base. Higher level relations as well as searching and inferential capabilities are normally encoded as special purpose programs. For future reference, we shall refer to these as "program-based systems." (For examples see References 7, 14, 17, 21, 24 or 25.) In general, logic-based systems offer more powerful inferential capabilities and permit input data using variables and containing quantifiers. Normally the addition of such deduction rules in a program-based system entails modifying existing programs or adding new routines to the system. However, logic-based systems currently lack capabilities for handling higher order logics (thus making it difficult to handle many features such as a "How many" type question), and in general their inability to deal with relevance makes them impractical for other than very small data bases consisting of a limited number of deduction rules. On the other hand, program based systems, by their nature, allow some degree of relevance to be incorporated into the data structures as well as the routines for searching these structures, at the expense of some degree of generality, and hence can be made to operate more efficiently on larger data bases. Furthermore it is difficult in either of these approaches to incorporate linguistic data into their data structures to provide a natural language capability with the system. As a result a natural language capability is usually provided by a separate component in the system which makes it difficult to allow the development and expansion of the subset of natural language in conjunction with the data base.

We feel that use of a heuristic problem solving language such as GOL as a basis for the semantic organization will permit the incorporation of the advantages of each of the above systems. The semantic representation used in ENGOLISH is a data structure (hereafter referred to as the DELTA notation) which is an extension of the GOL extensional and intensional data structures. The extensional data structures in GOL enable the definition of n-ary relations by an explicit encoding of the appropriate n-tuples. Alternatively, patterns containing free variables may be used to describe classes of n-tuples in compact form. The intensional data structures are λ-expressions which provide for the definition of one relation in terms of others; such definitions may be recursive. (For further details see Reference 13.) The DELTA notation combines these two into a single data structure also allowing provisions for representing negative data. In addition, it provides the capability for having string variables, as well as free variables; this makes the DELTA notation ideal for expressing grammars as well. The syntax for the DELTA notation is given in Figure 1 below.

As an example, in GOL one might code a maze solving routine by defining a GOL intensional data structure such as SOLVE-MAZE given below.

```
SOLVE-MAZE

 (S0000 (LAMBDA (X Y)
        (OR (CONNECTED X Y)
             (EXISTS (Z) (AND
                    (CONNECTED X Z)
                    (SOLVE-MAZE Z Y))))))
```

Then a maze such as the trivial example below



may be encoded with a GOL extensional data structure CONNECTED such as follows.

```
CONNECTED

 (SOOOO (EXT  (NIL)
              (START A)
              (A  B)
              (A  C)
              (B  D)
              (D  END)))
```

Alternatively, one could combine the two and use the single DELTA definition MAZE given below.

```
MAZE

 (SOOOO (DELTA  ((X Y))
                (T  (START A) T)
                (T  (A B        T)
                (T  (A C)       T)
                (T  (B D)  T)
                (T  (D END)  T)
                (T  (X Y)  (EXISTS (Z)
                  (AND (MAZE X Z)
                          (MAZE Z Y))))))
```

Here X and Y are defined as free variables and the first five "delta entries" encode the extension of MAZE (equivalent to CONNECTED defined above) and the last delta entry its intension (equivalent to SOLVE-

MAZE given above). The first delta entry

```
                (T   (START A)   T)
```

encodes the fact that MAZE is true of the 2-tuple (START A) under all conditions, and the last delta entry

```
(T  (X Y)  (EXISTS  (Z)
                    (AND (MAZE X Z)
                         (MAZE Z Y))))
```

encodes the fact that MAZE is true of any 2-tuple under the condition that there is a path connecting them.

Thus with respect to defining the "meaning" of natural language concepts, DELTA allows one to have an extensional as well as an intensional part as proposed by Carnap [3] in his method of extension and intension.

Thus

```
UNCLE
(SOOOO (DELTA ((X Y))
              (T     (HARRY   BILL) T)
              (F     (JOHN   SALLY) T)
              (T     (X Y) (EXISTS (Z)
         (AND (PARENT Z Y)
           (OR (BROTHER X Z)
              (BROTHER-IN-LAW X Z)))))
              (T     (X)     (EXISTS (Z)
           (UNCLE X Z)))))
```

would encode

(1) Harry is an uncle of Bill.
(2) John is not an uncle of Sally.
(3) X is an uncle of Y if X is a brother or brother-in-law of a parent of Y.
(4) X is an uncle if X is an uncle of someone.

with (1) and (2) being extensional and (3) and (4) expressing the intension of the binary relation "uncle" and unary relation "uncle" respectively.

GOL data structures such as above may subsequently be used to evaluate GOL-expressions which contain them. This evaluation may consist of testing the truth value for atomic arguments or generating values for arguments consisting of unbound variables. Thus

```
(LAMBDA ( ) (UNCLE (QUOTE HARRY)))
```

could represent the question

"Is Harry an uncle?"

and if applied to the above data structure UNCLE would result in a value of T. Similarly the GOL expression

```
(LAMBDA (A) (UNCLE A (QUOTE BILL)))
```

could represent the question

"Who is an uncle of BILL?"

and if applied to the data structure UNCLE would generate instances for the lambda-variable A.

To briefly illustrate the representation of a grammar, consider the following trivial example which defines the positive binary integers and which is equivalent to the context free grammar given in BNF below.

```
⟨positive-binary-integer⟩ : : =
                    0 | 1 | ⟨positive-binary-integer⟩ 0 |
                          | ⟨positive-binary-integer⟩ 1
```

```
POSITIVE-BINARY-INTEGER

(SOOOO (DELTA(  ( )  $X)
              (T         (0)      T)
              (T         (1)      T)
              (T         ($X 0)
 (POSITIVE-BINARY-INTEGER (QUOTE ($X))
              (T         ($X 1)
 (POSITIVE-BINARY-INTEGER (QUOTE ($X))
              (T         ($X)     T)
```

$X is defined as a string variable and the last entry above may be thought of as a completeness statement i.e., no other strings are positive binary integers.

DISCUSSION AND EXAMPLES

Although ENGOLISH is applicable to defining and interrogating various types of data bases, its development was guided by the desirability of building an interrogation system around physiological models, in particular a neural model which is under development as a means of representing facts concerning pharmacological data.

The first problem in this development was the choosing of some subset of English which was felt to be adequate for expressing questions concerning the information represented in such models. Because of the structural similarities between these models and pictures of simple objects, we began with a subset of English similar to that given in Kochen[9] and pioneered by the work of Kirsch,[7] concerning grammars for talking about pictures. Since any grammar for a collection of ques-

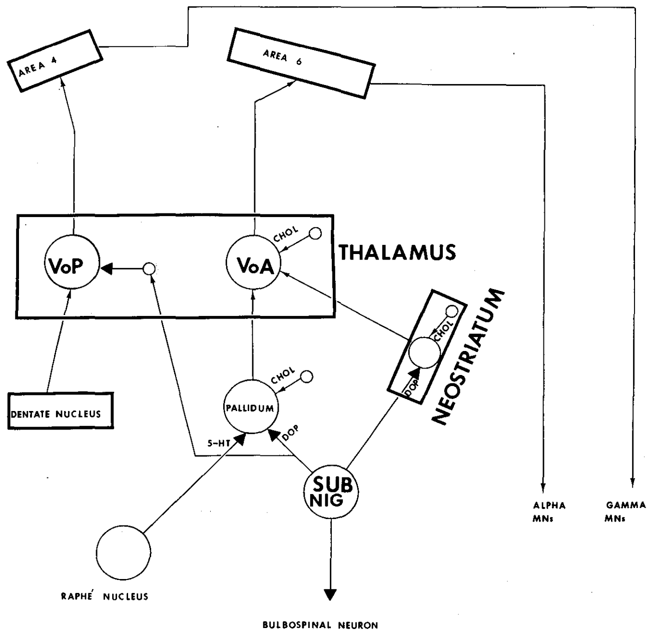


Figure 2

tions also entails as a subset a grammar for statements, it was decided that the defining of relations using such statements would also be a useful feature, thus allowing the construction of the data base itself using natural language. Since the language into which we translate such questions, i.e., GOL expressions is the same language in which the data base is structured, this was possible.

Illustrative example

As an illustrative example assume we wish to define and interrogate the prototype of a simple neural model given in Figure 2. The statements given in Figure 3 were used to define this data base in ENGOLISH.

Schema of Neural Structures and Their Transmitter Mechanisms for Figure 2.

VOP: Nucleus Ventralis Oralis Posterior
VOA: Nucleus Ventralis Oralis Anterior
ALPHA MNs and GAMMA MNs represent their respective motorneurons
AREA 4 and AREA 6 are the cytoarchitectural divisions of Motor Cortex
CHOL: cholinergic transmitter mechanism
DOP: dopaminergic transmitter mechanism
5-HT: serotonergic transmitter mechanism

Since ENGOLISH requires no prior knowledge concerning the given data base, i.e., the basic vocabulary of objects and names of relations, the model builder should enrich the vocabulary of the system for relations requiring more than one English word by using the SUBSTITUTE statement as illustrated in lines 00001 and 00003 above or by using a hyphen in the first appearance of the phrase as with "connected-inhibitory" in line 00005.

Using the small data base created by Figure 3, Figure 4 below illustrates types of questions which could be asked of it using ENGOLISH. (Lines preceded by * illustrate questions asked and the line (s) following the response from ENGOLISH.)

Note in the preceding that following "what-questions" the command "ANOTHER" may be used to elicit further instances for the "what-question" from ENGOLISH. Similarly, "REST" may be used for all remaining answers.

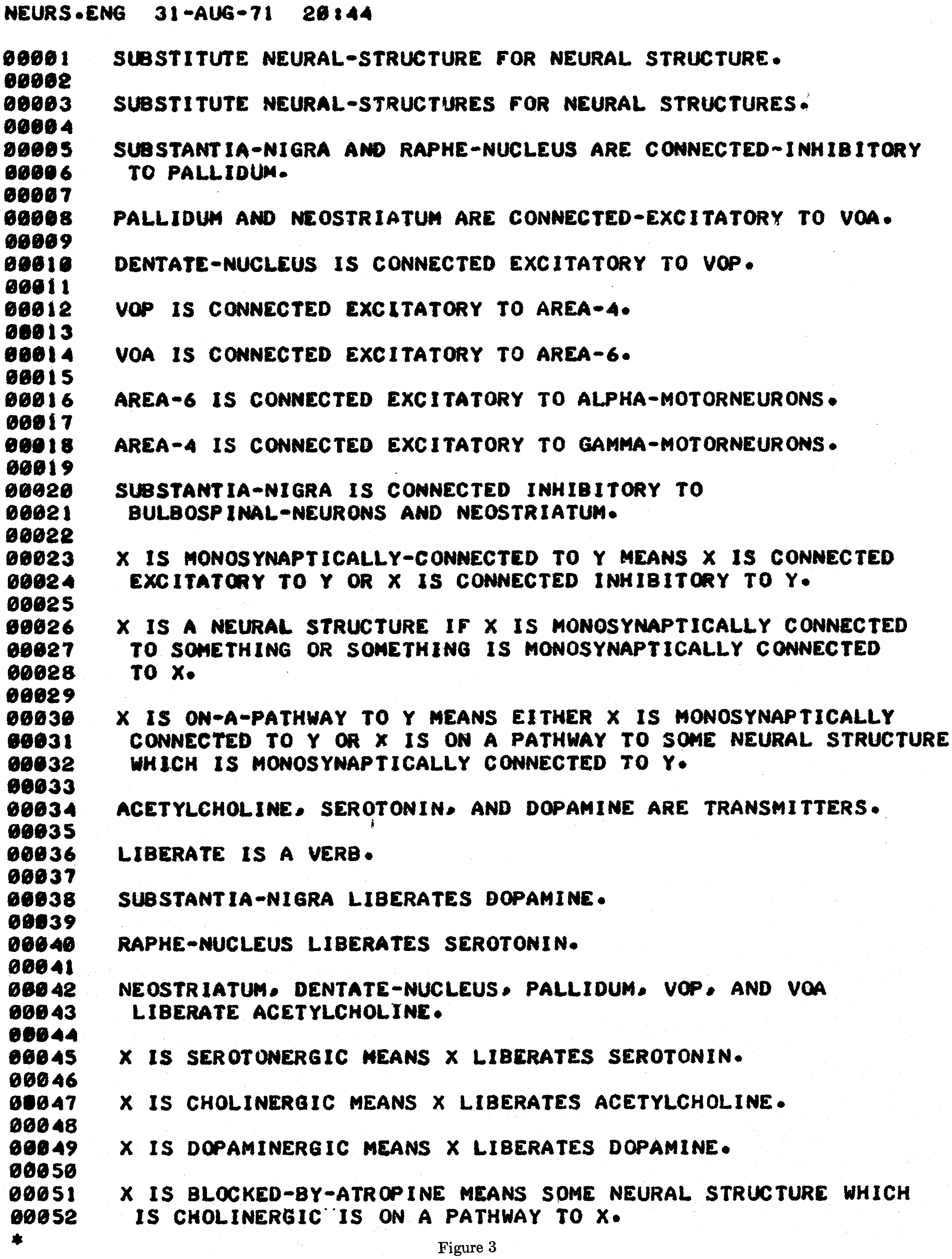
Encoding of literature

In addition to the information encoded within models, another important source of information is from literature which is relevant to such models. These could be used not only to cite justification for the structure of a given model by the model builder but also to contain published laboratory data which may corroborate or contradict results obtained when using such models.

To accommodate this type of data we decided that it should be encoded within the same data structures so that the same processors which search for answers to questions within a model, could also be used to answer such questions from the literature. Furthermore, the user should have the option of deciding to which source he wishes his interrogations directed—to a model, the literature or both.

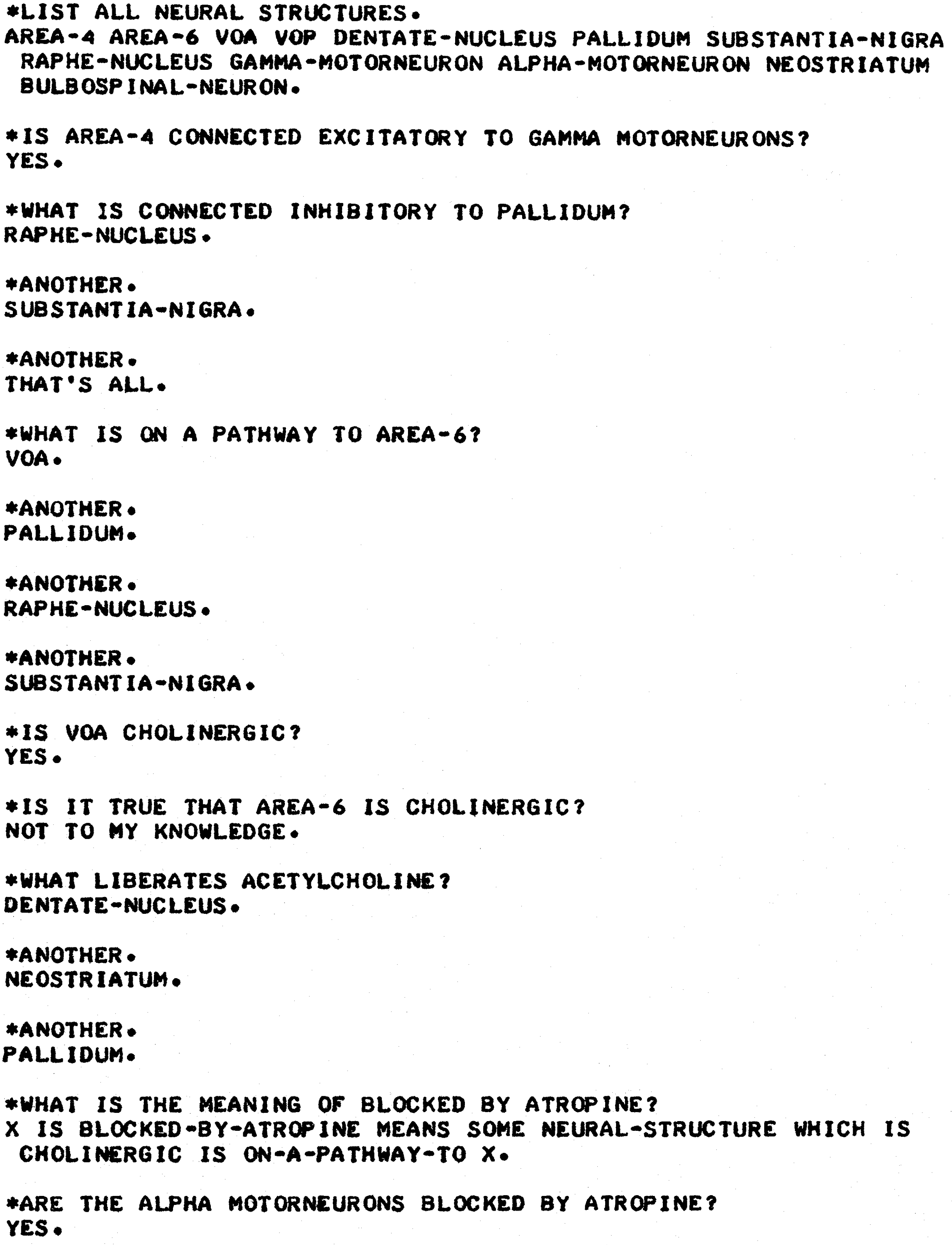
As a result, we currently envision the top level structure of this data base as a hierarchic state space. This is feasible because of GOL's existing capability for dealing with state-spaces in heuristic problem solving. The user then focuses attention on some node of this state space by the statement (LETS DISCUSS —) or by including the phrase (. . . ACCORDING TO X . . .) within a statement. Having done so, he then restricts the context to that state and any states accessible from it.

The indexing of data by states also provides the user a means of indexing his data base so that questions may be directed to specific parts of the data base, thus saving substantial searching time.



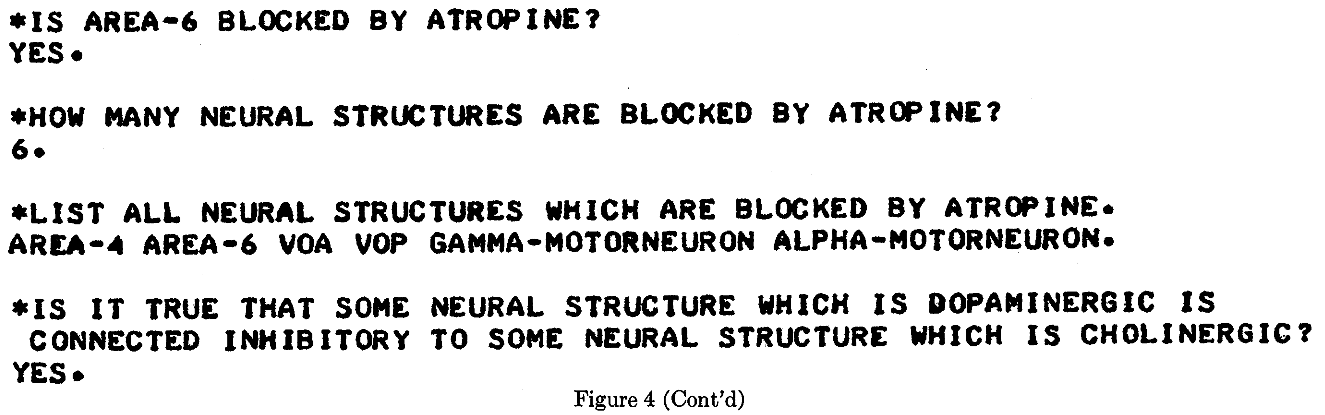
```
NEURS.ENG  31-AUG-71  20:44

SUBSTITUTE NEURAL-STRUCTURE FOR NEURAL STRUCTURE.

SUBSTITUTE NEURAL-STRUCTURES FOR NEURAL STRUCTURES.

SUBSTANTIA-NIGRA AND RAPHE-NUCLEUS ARE CONNECTED-INHIBITORY
 TO PALLIDUM.

PALLIDUM AND NEOSTRIATUM ARE CONNECTED-EXCITATORY TO VOA.

DENTATE-NUCLEUS IS CONNECTED EXCITATORY TO VOP.

VOP IS CONNECTED EXCITATORY TO AREA-4.

VOA IS CONNECTED EXCITATORY TO AREA-6.

AREA-6 IS CONNECTED EXCITATORY TO ALPHA-MOTORNEURONS.

AREA-4 IS CONNECTED EXCITATORY TO GAMMA-MOTORNEURONS.

SUBSTANTIA-NIGRA IS CONNECTED INHIBITORY TO
 BULBOSPINAL-NEURONS AND NEOSTRIATUM.

X IS MONOSYNAPTICALLY-CONNECTED TO Y MEANS X IS CONNECTED
 EXCITATORY TO Y OR X IS CONNECTED INHIBITORY TO Y.

X IS A NEURAL STRUCTURE IF X IS MONOSYNAPTICALLY CONNECTED
 TO SOMETHING OR SOMETHING IS MONOSYNAPTICALLY CONNECTED
 TO X.

X IS ON-A-PATHWAY TO Y MEANS EITHER X IS MONOSYNAPTICALLY
 CONNECTED TO Y OR X IS ON A PATHWAY TO SOME NEURAL STRUCTURE
 WHICH IS MONOSYNAPTICALLY CONNECTED TO Y.

ACETYLCHOLINE, SEROTONIN, AND DOPAMINE ARE TRANSMITTERS.

LIBERATE IS A VERB.

SUBSTANTIA-NIGRA LIBERATES DOPAMINE.

RAPHE-NUCLEUS LIBERATES SEROTONIN.

NEOSTRIATUM, DENTATE-NUCLEUS, PALLIDUM, VOP, AND VOA
 LIBERATE ACETYLCHOLINE.

X IS SEROTONERGIC MEANS X LIBERATES SEROTONIN.

X IS CHOLINERGIC MEANS X LIBERATES ACETYLCHOLINE.

X IS DOPAMINERGIC MEANS X LIBERATES DOPAMINE.

X IS BLOCKED-BY-ATROPINE MEANS SOME NEURAL STRUCTURE WHICH
 IS CHOLINERGIC IS ON A PATHWAY TO X.
*
```

Figure 3



```
*LIST ALL NEURAL STRUCTURES.
AREA-4 AREA-6 VOA VOP DENTATE-NUCLEUS PALLIDUM SUBSTANTIA-NIGRA
 RAPHE-NUCLEUS GAMMA-MOTORNEURON ALPHA-MOTORNEURON NEOSTRIATUM
 BULBOSPINAL-NEURON.

*IS AREA-4 CONNECTED EXCITATORY TO GAMMA MOTORNEURONS?
YES.

*WHAT IS CONNECTED INHIBITORY TO PALLIDUM?
RAPHE-NUCLEUS.

*ANOTHER.
SUBSTANTIA-NIGRA.

*ANOTHER.
THAT'S ALL.

*WHAT IS ON A PATHWAY TO AREA-6?
VOA.

*ANOTHER.
PALLIDUM.

*ANOTHER.
RAPHE-NUCLEUS.

*ANOTHER.
SUBSTANTIA-NIGRA.

*IS VOA CHOLINERGIC?
YES.

*IS IT TRUE THAT AREA-6 IS CHOLINERGIC?
NOT TO MY KNOWLEDGE.

*WHAT LIBERATES ACETYLCHOLINE?
DENTATE-NUCLEUS.

*ANOTHER.
NEOSTRIATUM.

*ANOTHER.
PALLIDUM.

*WHAT IS THE MEANING OF BLOCKED BY ATROPINE?
X IS BLOCKED-BY-ATROPINE MEANS SOME NEURAL-STRUCTURE WHICH IS
 CHOLINERGIC IS ON-A-PATHWAY-TO X.

*ARE THE ALPHA MOTORNEURONS BLOCKED BY ATROPINE?
YES.
```

Figure 4



```
*IS AREA-6 BLOCKED BY ATROPINE?
YES.

*HOW MANY NEURAL STRUCTURES ARE BLOCKED BY ATROPINE?
6.

*LIST ALL NEURAL STRUCTURES WHICH ARE BLOCKED BY ATROPINE.
AREA-4 AREA-6 VOA VOP GAMMA-MOTORNEURON ALPHA-MOTORNEURON.

*IS IT TRUE THAT SOME NEURAL STRUCTURE WHICH IS DOPAMINERGIC IS
 CONNECTED INHIBITORY TO SOME NEURAL STRUCTURE WHICH IS CHOLINERGIC?
YES.
```

Figure 4 (Cont'd)

A complete example

Within the framework of the previous section one may include facts from the literature into such a data base by expressing them as ENGOLISH statements as represented by the example in Figure 5.

In a similar manner, the model builder may encode comments about his model. Such data is useful not only for future reference by the model builder but also by future users of such models to explain and justify aspects of them.

Having encoded such data for a neural model and several documents relevant to it, one may then interrogate this data base with queries such as illustrated in Figure 6.

ENGOLISH IMPLEMENTATION

The current version of ENGOLISH is implemented in GOL enriched LISP. The translator consists of a top-to-bottom parser utilizing heuristics on syntactical patterns as well as the data base itself to aid in the recognition of sentences, questions, names, etc. These patterns are defined by GOL DELTA data structures and the parser and heuristics by GOL intensional data structures and LISP subroutines. As a result it resembles the conceptual parser described by Shank[19,20] as opposed to totally grammar based systems. The parsing algorithm places stress on "function words" such as articles, prepositions, quantifiers etc., and as a consequence is capable of parsing sentences containing nouns, verbs, and adjectives which were not previously defined within the data base and for which no syntactic information is available. Such words which are defined in the data base are used to aid in the parsing and disambiguating of sentences.

Having parsed an English statement or query, the phrase marker generated is used to generate the equivalent GOL expression. This expression is then evaluated, and if a statement, it will result in the appropriate data structure being defined or modified, and if a question, in the appropriate value being generated or retrieved, to provide an answer to the question.

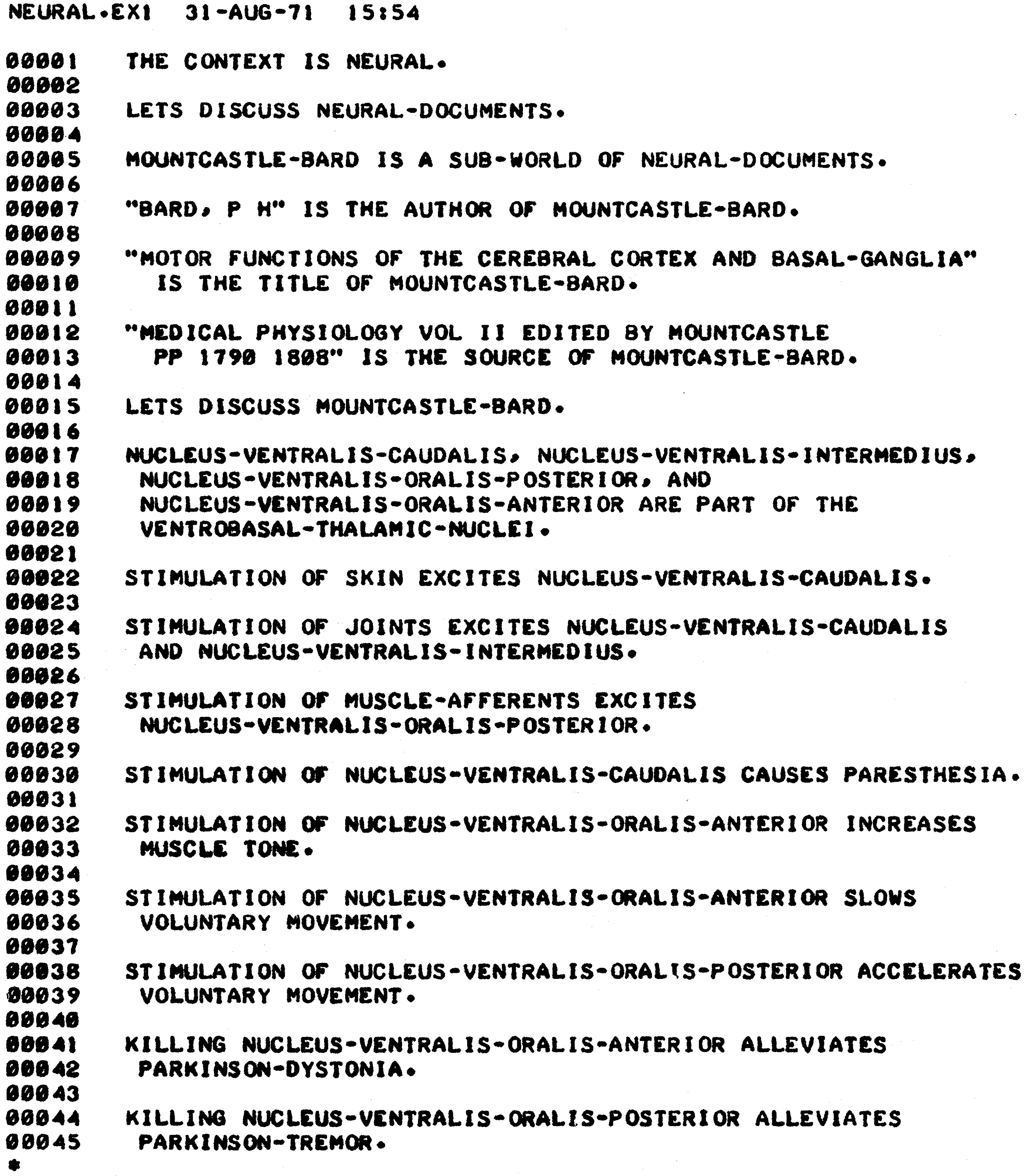
DISCUSSION AND FUTURE DEVELOPMENTS

The preceding sections described the current status of ENGOLISH. Even though it has convinced us of the feasibility of translating from a subset of English to GOL and of its desirability as a basis for a query system, there still remains much to be done.

Currently, steps are being taken to expand the conversational capabilities of ENGOLISH. In addition to expanding the syntax, there is need for more interaction between man and machine. In addition to utilizing its own data base to answer questions, ENGOLISH's utility would be increased by adding capabilities for it to generate sub-questions to the user, i.e., to also make use of the users knowledge in answering his questions. For example, consider a true-false question such as IS SUBSTANTIA-NIGRA BLOCKED BY ATROPINE? This question would currently result in a "YES," "NO," or "NOT TO MY KNOWLEDGE" response from ENGOLISH, depending on the success of searching out pathways to Substantia-Nigra for a neural structure which is cholinergic (see Figure 3). If none were found, but neural structures were tested for which no transmitter was known, it would be desirable to have the system query the user with

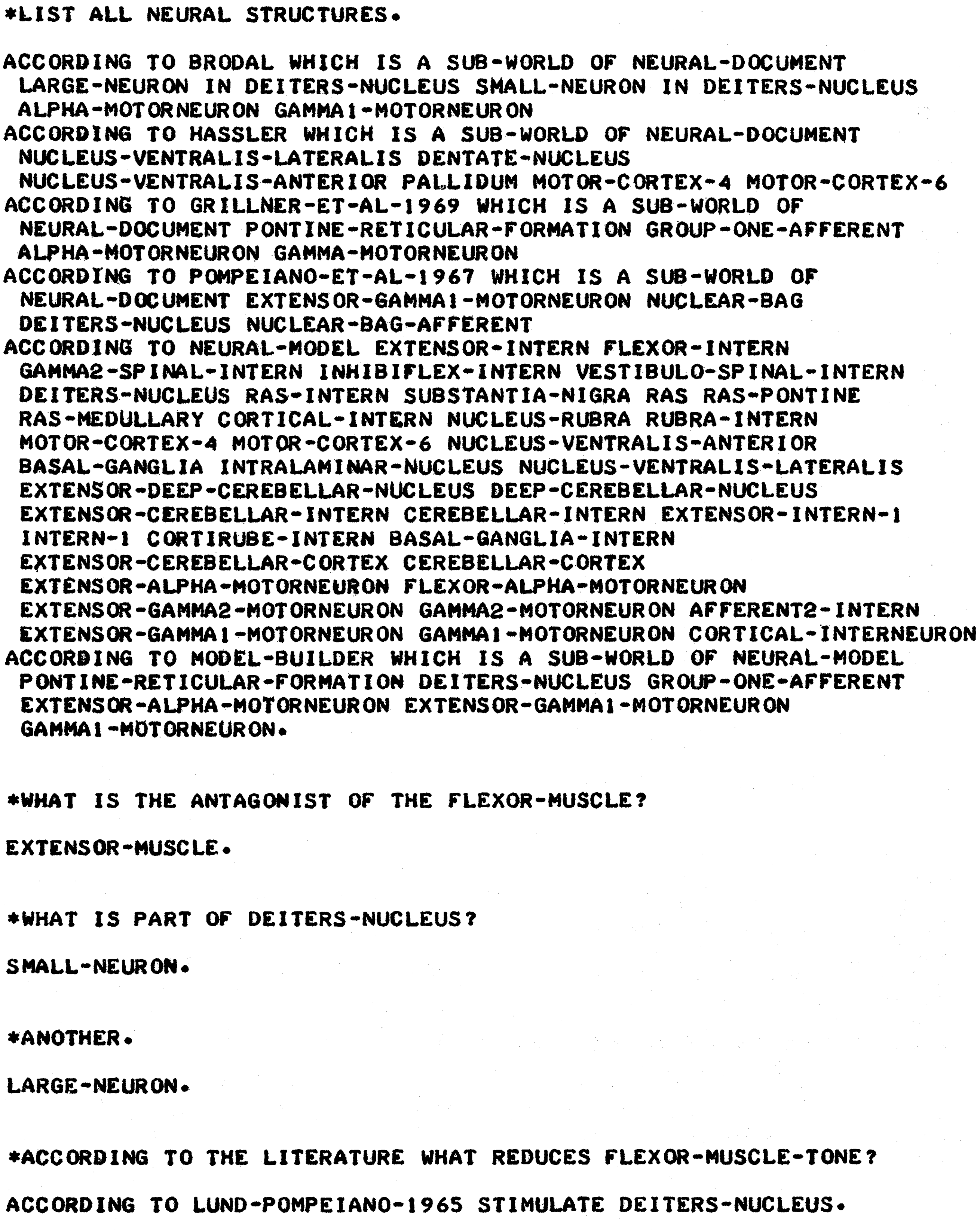
IS _____ CHOLINERGIC?
or
WHAT DOES _____ LIBERATE?

as opposed to merely responding "NOT TO MY KNOWLEDGE."



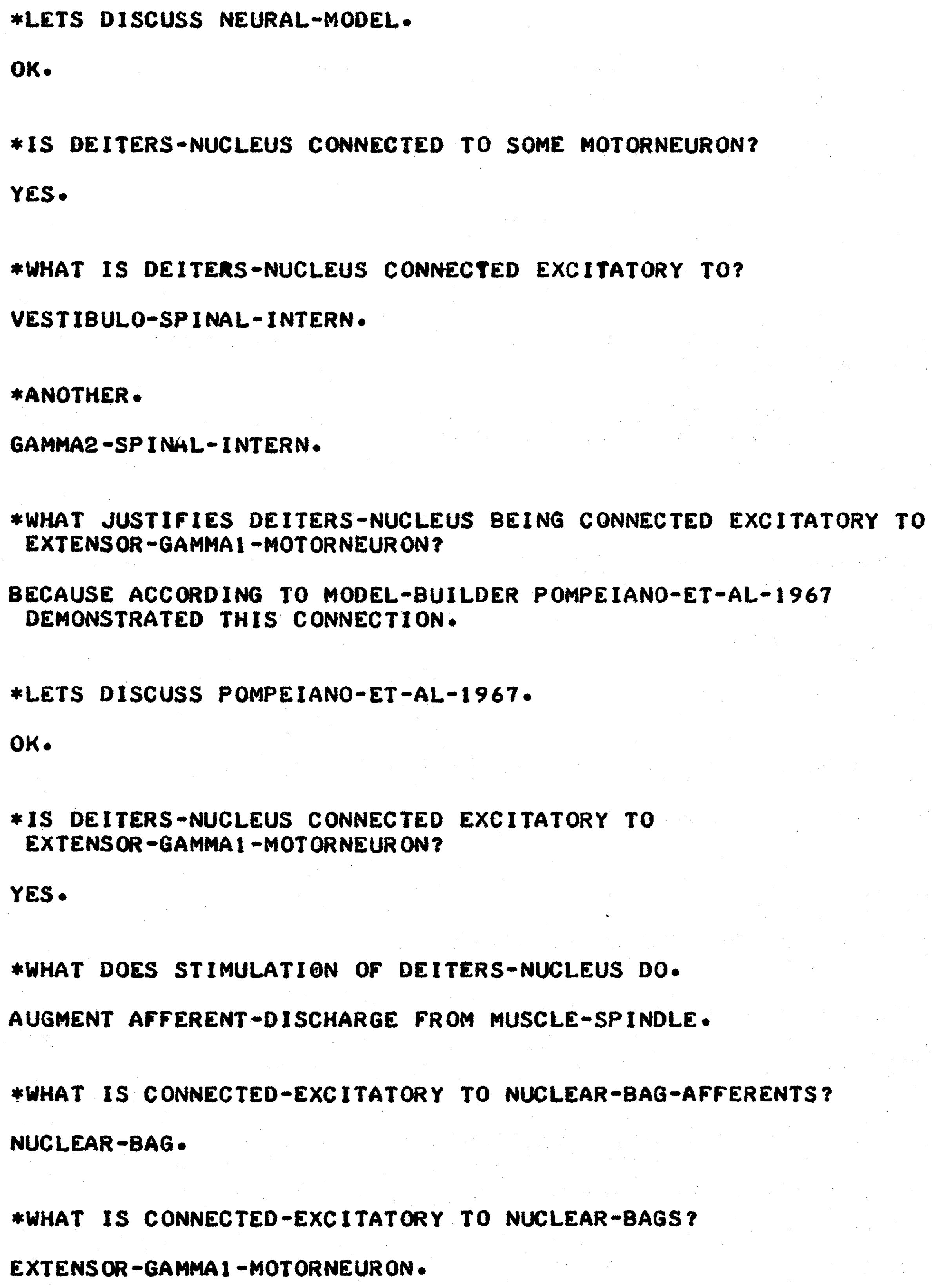
```
NEURAL.EX1  31-AUG-71  15:54

THE CONTEXT IS NEURAL.

LETS DISCUSS NEURAL-DOCUMENTS.

MOUNTCASTLE-BARD IS A SUB-WORLD OF NEURAL-DOCUMENTS.

"BARD, P H" IS THE AUTHOR OF MOUNTCASTLE-BARD.

"MOTOR FUNCTIONS OF THE CEREBRAL CORTEX AND BASAL-GANGLIA"
  IS THE TITLE OF MOUNTCASTLE-BARD.

"MEDICAL PHYSIOLOGY VOL II EDITED BY MOUNTCASTLE
  PP 1790 1808" IS THE SOURCE OF MOUNTCASTLE-BARD.

LETS DISCUSS MOUNTCASTLE-BARD.

NUCLEUS-VENTRALIS-CAUDALIS, NUCLEUS-VENTRALIS-INTERMEDIUS,
 NUCLEUS-VENTRALIS-ORALIS-POSTERIOR, AND
 NUCLEUS-VENTRALIS-ORALIS-ANTERIOR ARE PART OF THE
 VENTROBASAL-THALAMIC-NUCLEI.

STIMULATION OF SKIN EXCITES NUCLEUS-VENTRALIS-CAUDALIS.

STIMULATION OF JOINTS EXCITES NUCLEUS-VENTRALIS-CAUDALIS
 AND NUCLEUS-VENTRALIS-INTERMEDIUS.

STIMULATION OF MUSCLE-AFFERENTS EXCITES
 NUCLEUS-VENTRALIS-ORALIS-POSTERIOR.

STIMULATION OF NUCLEUS-VENTRALIS-CAUDALIS CAUSES PARESTHESIA.

STIMULATION OF NUCLEUS-VENTRALIS-ORALIS-ANTERIOR INCREASES
 MUSCLE TONE.

STIMULATION OF NUCLEUS-VENTRALIS-ORALIS-ANTERIOR SLOWS
 VOLUNTARY MOVEMENT.

STIMULATION OF NUCLEUS-VENTRALIS-ORALIS-POSTERIOR ACCELERATES
 VOLUNTARY MOVEMENT.

KILLING NUCLEUS-VENTRALIS-ORALIS-ANTERIOR ALLEVIATES
 PARKINSON-DYSTONIA.

KILLING NUCLEUS-VENTRALIS-ORALIS-POSTERIOR ALLEVIATES
 PARKINSON-TREMOR.
*
```

Figure 5



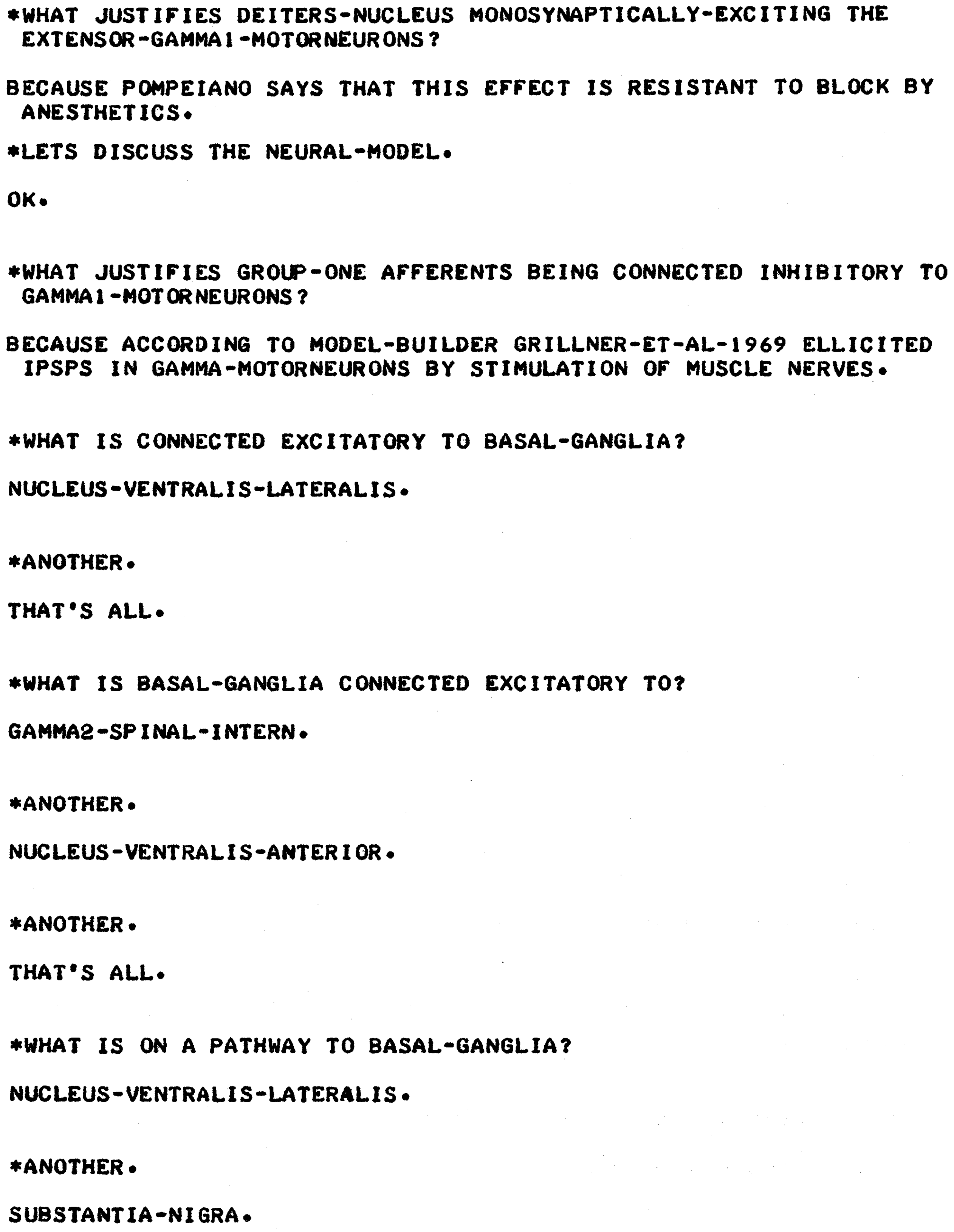
```
*LIST ALL NEURAL STRUCTURES.

ACCORDING TO BRODAL WHICH IS A SUB-WORLD OF NEURAL-DOCUMENT
 LARGE-NEURON IN DEITERS-NUCLEUS SMALL-NEURON IN DEITERS-NUCLEUS
 ALPHA-MOTORNEURON GAMMA1-MOTORNEURON
ACCORDING TO HASSLER WHICH IS A SUB-WORLD OF NEURAL-DOCUMENT
 NUCLEUS-VENTRALIS-LATERALIS DENTATE-NUCLEUS
 NUCLEUS-VENTRALIS-ANTERIOR PALLIDUM MOTOR-CORTEX-4 MOTOR-CORTEX-6
ACCORDING TO GRILLNER-ET-AL-1969 WHICH IS A SUB-WORLD OF
 NEURAL-DOCUMENT PONTINE-RETICULAR-FORMATION GROUP-ONE-AFFERENT
 ALPHA-MOTORNEURON GAMMA-MOTORNEURON
ACCORDING TO POMPEIANO-ET-AL-1967 WHICH IS A SUB-WORLD OF
 NEURAL-DOCUMENT EXTENSOR-GAMMA1-MOTORNEURON NUCLEAR-BAG
 DEITERS-NUCLEUS NUCLEAR-BAG-AFFERENT
ACCORDING TO NEURAL-MODEL EXTENSOR-INTERN FLEXOR-INTERN
 GAMMA2-SPINAL-INTERN INHIBIFLEX-INTERN VESTIBULO-SPINAL-INTERN
 DEITERS-NUCLEUS RAS-INTERN SUBSTANTIA-NIGRA RAS RAS-PONTINE
 RAS-MEDULLARY CORTICAL-INTERN NUCLEUS-RUBRA RUBRA-INTERN
 MOTOR-CORTEX-4 MOTOR-CORTEX-6 NUCLEUS-VENTRALIS-ANTERIOR
 BASAL-GANGLIA INTRALAMINAR-NUCLEUS NUCLEUS-VENTRALIS-LATERALIS
 EXTENSOR-DEEP-CEREBELLAR-NUCLEUS DEEP-CEREBELLAR-NUCLEUS
 EXTENSOR-CEREBELLAR-INTERN CEREBELLAR-INTERN EXTENSOR-INTERN-1
 INTERN-1 CORTIRUBE-INTERN BASAL-GANGLIA-INTERN
 EXTENSOR-CEREBELLAR-CORTEX CEREBELLAR-CORTEX
 EXTENSOR-ALPHA-MOTORNEURON FLEXOR-ALPHA-MOTORNEURON
 EXTENSOR-GAMMA2-MOTORNEURON GAMMA2-MOTORNEURON AFFERENT2-INTERN
 EXTENSOR-GAMMA1-MOTORNEURON GAMMA1-MOTORNEURON CORTICAL-INTERNEURON
ACCORDING TO MODEL-BUILDER WHICH IS A SUB-WORLD OF NEURAL-MODEL
 PONTINE-RETICULAR-FORMATION DEITERS-NUCLEUS GROUP-ONE-AFFERENT
 EXTENSOR-ALPHA-MOTORNEURON EXTENSOR-GAMMA1-MOTORNEURON
 GAMMA1-MOTORNEURON.

*WHAT IS THE ANTAGONIST OF THE FLEXOR-MUSCLE?

EXTENSOR-MUSCLE.

*WHAT IS PART OF DEITERS-NUCLEUS?

SMALL-NEURON.

*ANOTHER.

LARGE-NEURON.

*ACCORDING TO THE LITERATURE WHAT REDUCES FLEXOR-MUSCLE-TONE?

ACCORDING TO LUND-POMPEIANO-1965 STIMULATE DEITERS-NUCLEUS.
```

Figure 6



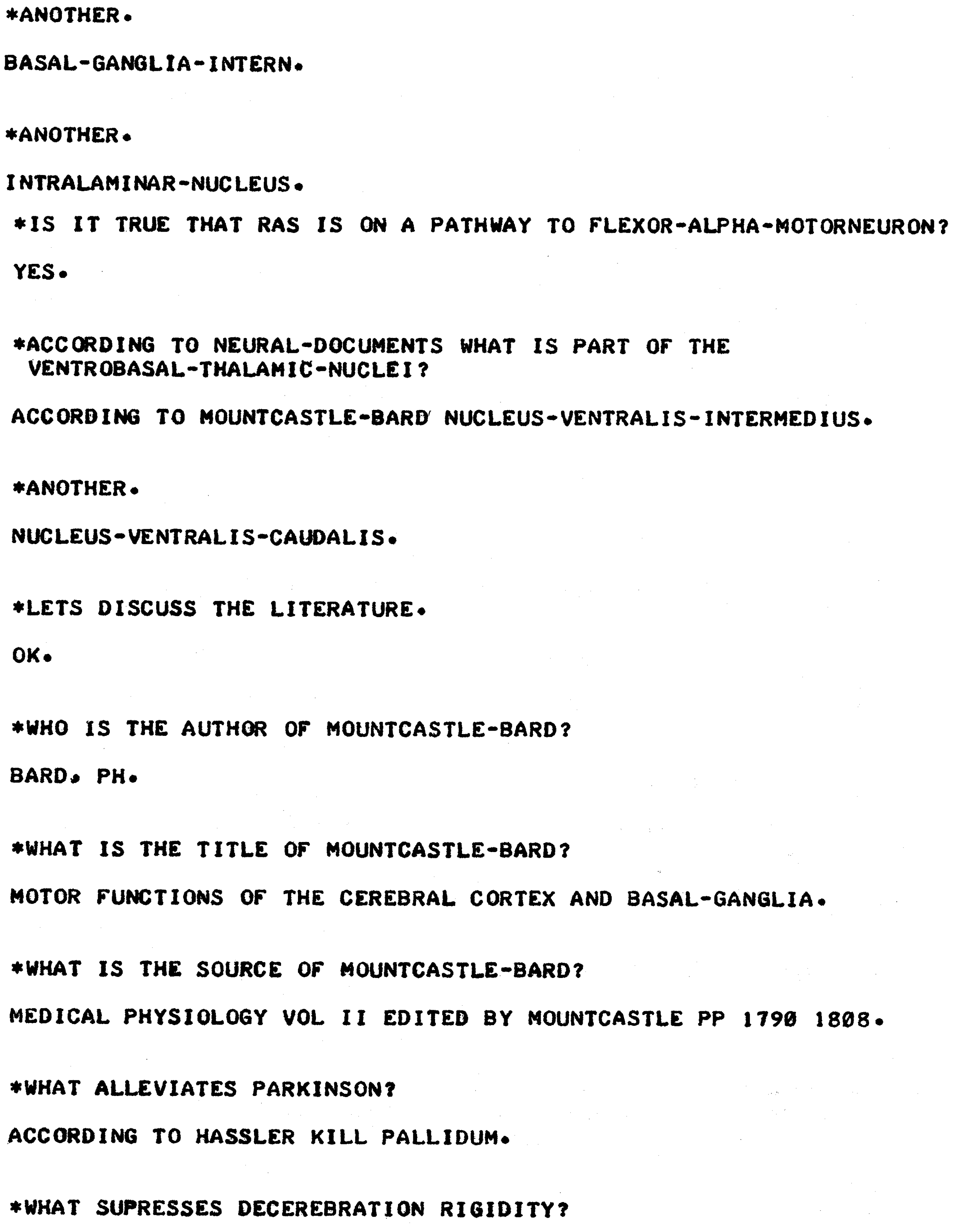
```
*LETS DISCUSS NEURAL-MODEL.

OK.


*IS DEITERS-NUCLEUS CONNECTED TO SOME MOTORNEURON?

YES.


*WHAT IS DEITERS-NUCLEUS CONNECTED EXCITATORY TO?

VESTIBULO-SPINAL-INTERN.


*ANOTHER.

GAMMA2-SPINAL-INTERN.


*WHAT JUSTIFIES DEITERS-NUCLEUS BEING CONNECTED EXCITATORY TO
 EXTENSOR-GAMMA1-MOTORNEURON?

BECAUSE ACCORDING TO MODEL-BUILDER POMPEIANO-ET-AL-1967
 DEMONSTRATED THIS CONNECTION.


*LETS DISCUSS POMPEIANO-ET-AL-1967.

OK.


*IS DEITERS-NUCLEUS CONNECTED EXCITATORY TO
 EXTENSOR-GAMMA1-MOTORNEURON?

YES.


*WHAT DOES STIMULATION OF DEITERS-NUCLEUS DO.

AUGMENT AFFERENT-DISCHARGE FROM MUSCLE-SPINDLE.


*WHAT IS CONNECTED-EXCITATORY TO NUCLEAR-BAG-AFFERENTS?

NUCLEAR-BAG.


*WHAT IS CONNECTED-EXCITATORY TO NUCLEAR-BAGS?

EXTENSOR-GAMMA1-MOTORNEURON.
```

Figure 6 (Cont'd)



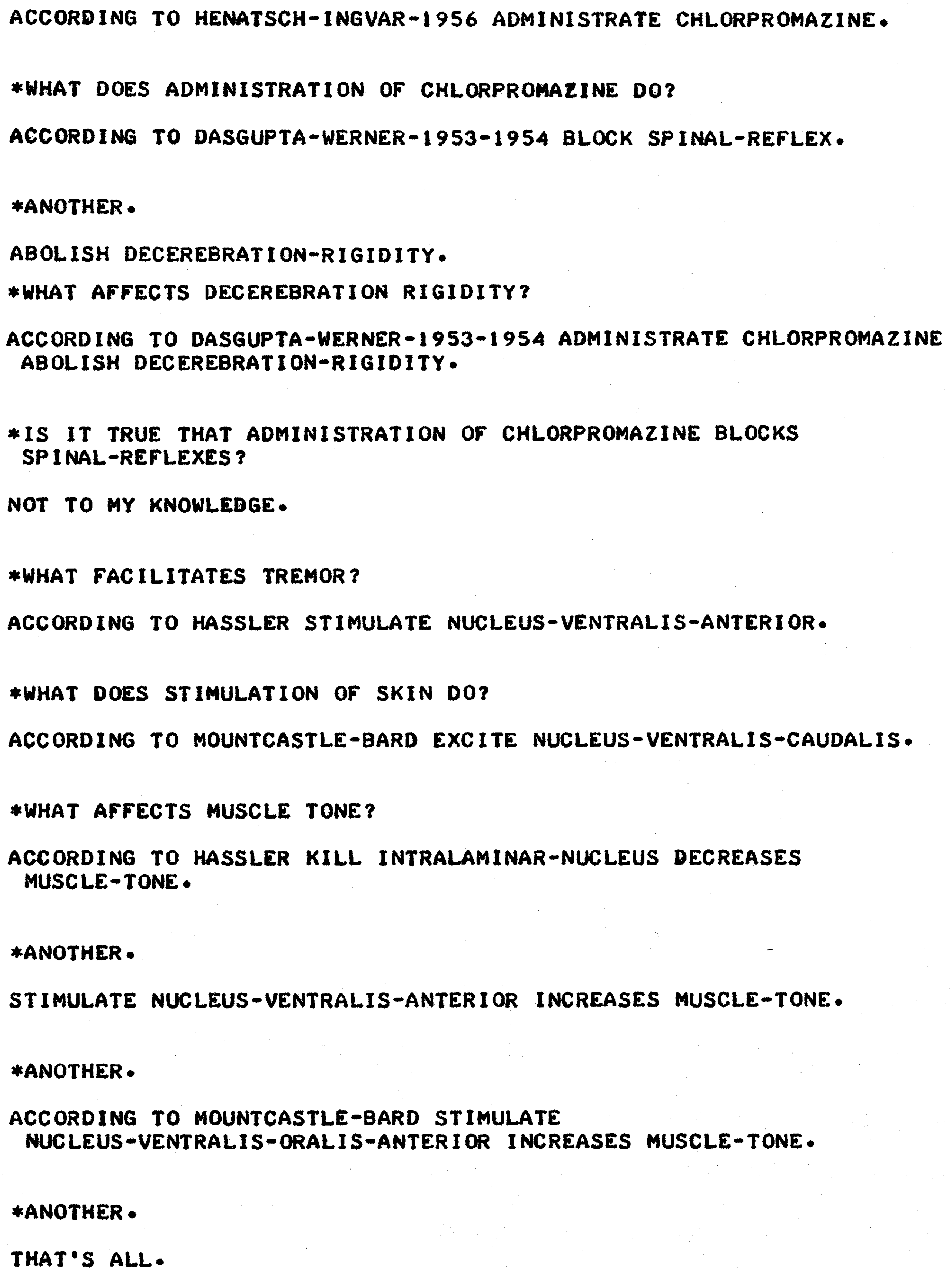
```
*WHAT JUSTIFIES DEITERS-NUCLEUS MONOSYNAPTICALLY-EXCITING THE
 EXTENSOR-GAMMA1-MOTORNEURONS?

BECAUSE POMPEIANO SAYS THAT THIS EFFECT IS RESISTANT TO BLOCK BY
 ANESTHETICS.

*LETS DISCUSS THE NEURAL-MODEL.

OK.


*WHAT JUSTIFIES GROUP-ONE AFFERENTS BEING CONNECTED INHIBITORY TO
 GAMMA1-MOTORNEURONS?

BECAUSE ACCORDING TO MODEL-BUILDER GRILLNER-ET-AL-1969 ELLICITED
 IPSPS IN GAMMA-MOTORNEURONS BY STIMULATION OF MUSCLE NERVES.


*WHAT IS CONNECTED EXCITATORY TO BASAL-GANGLIA?

NUCLEUS-VENTRALIS-LATERALIS.


*ANOTHER.

THAT'S ALL.


*WHAT IS BASAL-GANGLIA CONNECTED EXCITATORY TO?

GAMMA2-SPINAL-INTERN.


*ANOTHER.

NUCLEUS-VENTRALIS-ANTERIOR.


*ANOTHER.

THAT'S ALL.


*WHAT IS ON A PATHWAY TO BASAL-GANGLIA?

NUCLEUS-VENTRALIS-LATERALIS.


*ANOTHER.

SUBSTANTIA-NIGRA.
```

Figure 6 (Cont'd)



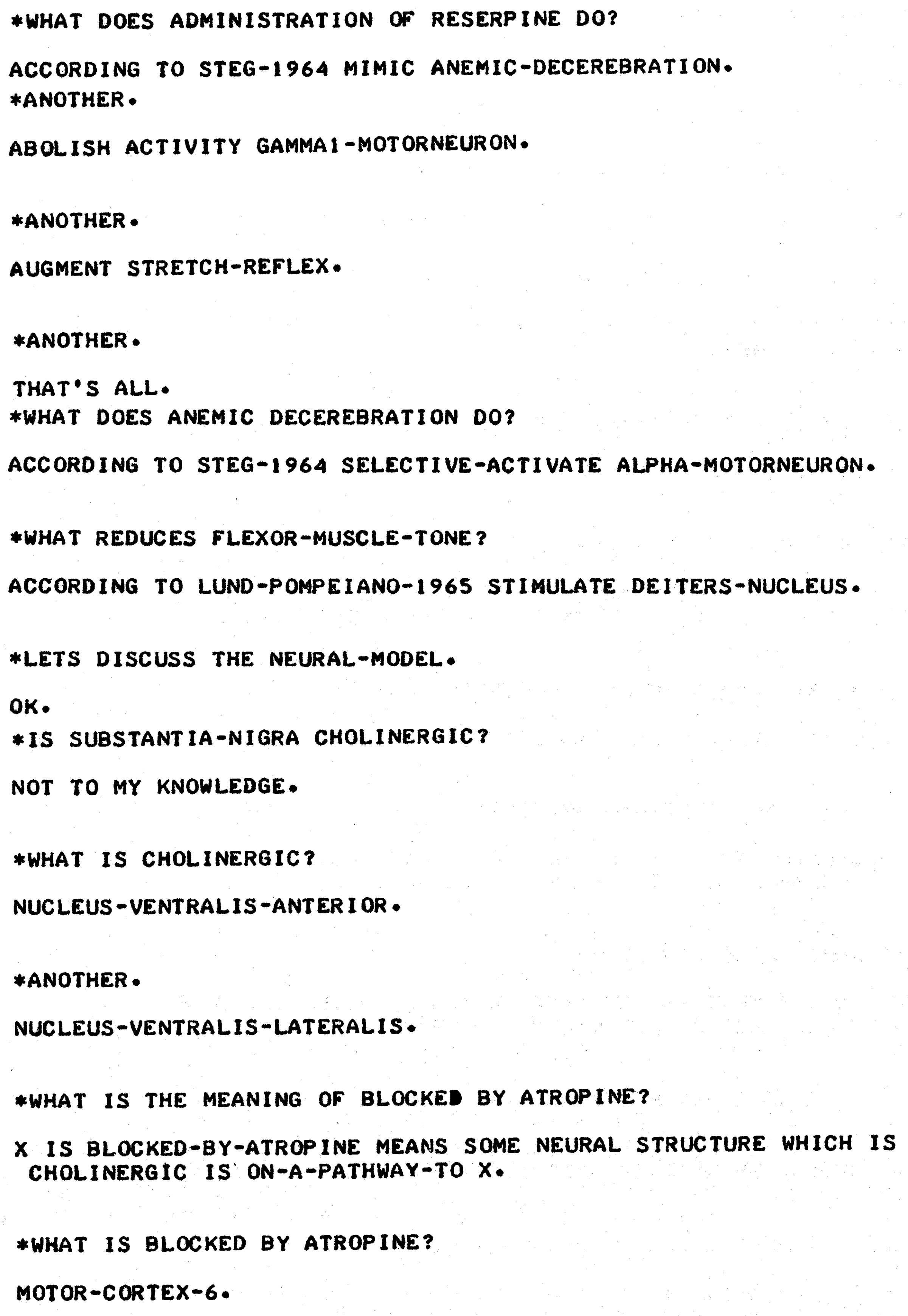
```
*ANOTHER.

BASAL-GANGLIA-INTERN.


*ANOTHER.

INTRALAMINAR-NUCLEUS.

 *IS IT TRUE THAT RAS IS ON A PATHWAY TO FLEXOR-ALPHA-MOTORNEURON?

 YES.


 *ACCORDING TO NEURAL-DOCUMENTS WHAT IS PART OF THE
  VENTROBASAL-THALAMIC-NUCLEI?

 ACCORDING TO MOUNTCASTLE-BARD' NUCLEUS-VENTRALIS-INTERMEDIUS.


 *ANOTHER.

 NUCLEUS-VENTRALIS-CAUDALIS.


 *LETS DISCUSS THE LITERATURE.

 OK.


 *WHO IS THE AUTHOR OF MOUNTCASTLE-BARD?

 BARD, PH.


 *WHAT IS THE TITLE OF MOUNTCASTLE-BARD?

 MOTOR FUNCTIONS OF THE CEREBRAL CORTEX AND BASAL-GANGLIA.


 *WHAT IS THE SOURCE OF MOUNTCASTLE-BARD?

 MEDICAL PHYSIOLOGY VOL II EDITED BY MOUNTCASTLE PP 1790 1808.


 *WHAT ALLEVIATES PARKINSON?

 ACCORDING TO HASSLER KILL PALLIDUM.


 *WHAT SUPRESSES DECEREBRATION RIGIDITY?
```

Figure 6 (Cont'd)



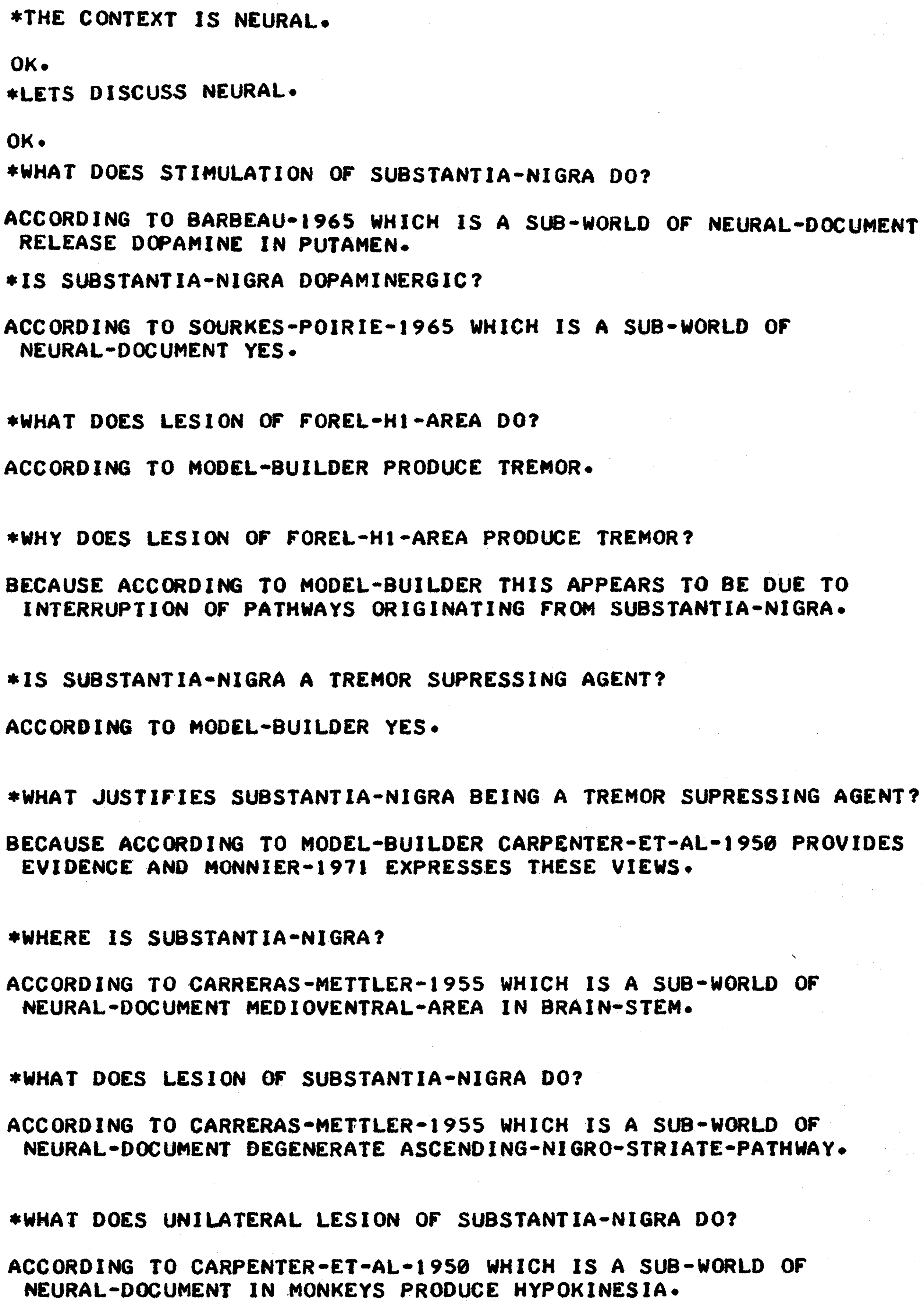
```
ACCORDING TO HENATSCH-INGVAR-1956 ADMINISTRATE CHLORPROMAZINE.


*WHAT DOES ADMINISTRATION OF CHLORPROMAZINE DO?

ACCORDING TO DASGUPTA-WERNER-1953-1954 BLOCK SPINAL-REFLEX.


*ANOTHER.

ABOLISH DECEREBRATION-RIGIDITY.
*WHAT AFFECTS DECEREBRATION RIGIDITY?

ACCORDING TO DASGUPTA-WERNER-1953-1954 ADMINISTRATE CHLORPROMAZINE
 ABOLISH DECEREBRATION-RIGIDITY.


*IS IT TRUE THAT ADMINISTRATION OF CHLORPROMAZINE BLOCKS
 SPINAL-REFLEXES?

NOT TO MY KNOWLEDGE.


*WHAT FACILITATES TREMOR?

ACCORDING TO HASSLER STIMULATE NUCLEUS-VENTRALIS-ANTERIOR.


*WHAT DOES STIMULATION OF SKIN DO?

ACCORDING TO MOUNTCASTLE-BARD EXCITE NUCLEUS-VENTRALIS-CAUDALIS.


*WHAT AFFECTS MUSCLE TONE?

ACCORDING TO HASSLER KILL INTRALAMINAR-NUCLEUS DECREASES
 MUSCLE-TONE.


*ANOTHER.

STIMULATE NUCLEUS-VENTRALIS-ANTERIOR INCREASES MUSCLE-TONE.


*ANOTHER.

ACCORDING TO MOUNTCASTLE-BARD STIMULATE
 NUCLEUS-VENTRALIS-ORALIS-ANTERIOR INCREASES MUSCLE-TONE.


*ANOTHER.

THAT'S ALL.
```

Figure 6 (Cont'd)



```
*WHAT DOES ADMINISTRATION OF RESERPINE DO?

ACCORDING TO STEG-1964 MIMIC ANEMIC-DECEREBRATION.
*ANOTHER.

ABOLISH ACTIVITY GAMMA1-MOTORNEURON.


*ANOTHER.

AUGMENT STRETCH-REFLEX.


*ANOTHER.

THAT'S ALL.
*WHAT DOES ANEMIC DECEREBRATION DO?

ACCORDING TO STEG-1964 SELECTIVE-ACTIVATE ALPHA-MOTORNEURON.


*WHAT REDUCES FLEXOR-MUSCLE-TONE?

ACCORDING TO LUND-POMPEIANO-1965 STIMULATE DEITERS-NUCLEUS.


*LETS DISCUSS THE NEURAL-MODEL.

OK.
*IS SUBSTANTIA-NIGRA CHOLINERGIC?

NOT TO MY KNOWLEDGE.


*WHAT IS CHOLINERGIC?

NUCLEUS-VENTRALIS-ANTERIOR.


*ANOTHER.

NUCLEUS-VENTRALIS-LATERALIS.


*WHAT IS THE MEANING OF BLOCKED BY ATROPINE?

X IS BLOCKED-BY-ATROPINE MEANS SOME NEURAL STRUCTURE WHICH IS
 CHOLINERGIC IS ON-A-PATHWAY-TO X.


*WHAT IS BLOCKED BY ATROPINE?

MOTOR-CORTEX-6.
```

Figure 6 (Cont'd)



```
*THE CONTEXT IS NEURAL.

OK.
*LETS DISCUSS NEURAL.

OK.
*WHAT DOES STIMULATION OF SUBSTANTIA-NIGRA DO?

ACCORDING TO BARBEAU-1965 WHICH IS A SUB-WORLD OF NEURAL-DOCUMENT
 RELEASE DOPAMINE IN PUTAMEN.
*IS SUBSTANTIA-NIGRA DOPAMINERGIC?

ACCORDING TO SOURKES-POIRIE-1965 WHICH IS A SUB-WORLD OF
 NEURAL-DOCUMENT YES.


*WHAT DOES LESION OF FOREL-H1-AREA DO?

ACCORDING TO MODEL-BUILDER PRODUCE TREMOR.


*WHY DOES LESION OF FOREL-H1-AREA PRODUCE TREMOR?

BECAUSE ACCORDING TO MODEL-BUILDER THIS APPEARS TO BE DUE TO
 INTERRUPTION OF PATHWAYS ORIGINATING FROM SUBSTANTIA-NIGRA.


*IS SUBSTANTIA-NIGRA A TREMOR SUPRESSING AGENT?

ACCORDING TO MODEL-BUILDER YES.


*WHAT JUSTIFIES SUBSTANTIA-NIGRA BEING A TREMOR SUPRESSING AGENT?

BECAUSE ACCORDING TO MODEL-BUILDER CARPENTER-ET-AL-1950 PROVIDES
 EVIDENCE AND MONNIER-1971 EXPRESSES THESE VIEWS.


*WHERE IS SUBSTANTIA-NIGRA?

ACCORDING TO CARRERAS-METTLER-1955 WHICH IS A SUB-WORLD OF
 NEURAL-DOCUMENT MEDIOVENTRAL-AREA IN BRAIN-STEM.


*WHAT DOES LESION OF SUBSTANTIA-NIGRA DO?

ACCORDING TO CARRERAS-METTLER-1955 WHICH IS A SUB-WORLD OF
 NEURAL-DOCUMENT DEGENERATE ASCENDING-NIGRO-STRIATE-PATHWAY.


*WHAT DOES UNILATERAL LESION OF SUBSTANTIA-NIGRA DO?

ACCORDING TO CARPENTER-ET-AL-1950 WHICH IS A SUB-WORLD OF
 NEURAL-DOCUMENT IN MONKEYS PRODUCE HYPOKINESIA.
```

Figure 6 (Cont'd)

Effectively, ENGOLISH could then answer questions or sub-questions by searching its data base, running dynamic models, or querying the user. Furthermore this capability could also be used to allow the system to ask questions concerning new data presented to it and perhaps eventually the grammar itself which is being used. Thus, given a user who did not know the organization of the data base, but who wished to encode a fact such as

NEURON-A IS CONNECTED TO NEURON-B

would result in the structure "CONNECTED" being defined or updated. Of more use would be the capability for the system to respond with

IS NEURON-A CONNECTED-EXCITATORY to NEURON-B

or

IS NEURON-A CONNECTED-INHIBITORY to NEURON-B

so that the new information could be represented more consistently with existing information.

Another addition consists of making the state transformation facilities of GOL available to the user of ENGOLISH. In addition to defining states the user could then define actions or transformations (such as drug administrations would entail), capable of creating new states from old ones. This should increase the utility of ENGOLISH for problem solving in general, since problem solving often involves changing some model, by using specified transformations or actions, in search of a desired goal state for the model.

Furthermore, we are also exploring the potential within the current system for the generation of meaningful discourse by machine. This would enable the generation of answers to some "how" and "why" questions, by generating text describing the path of the GOL heuristic search process. Thus, it would describe the sequence of transformations and their effects in a search for the goal state in a problem solving situation.

REFERENCES

1 N D BELNAP JR
An analysis of questions: Preliminary report
System Development Corporation Technical Memorandum TM-1287/000/00 June 1963

2 H G BOHNERT P O BACKER
Automatic English-to-logic translation in a simplified model—A study in the logic of grammar
IBM Watson Research Center RC-1744 Yorktown Heights New York January 11 1967

3 R CARNAP
Meaning and necessity
University of Chicago Press Chicago Illinois 1947

4 N CHOMSKY
Aspects of the theory of syntax
MIT Press Cambridge Massachusetts 1965

5 C GREEN
Theorem proving by resolution as a basis for question-answering systems
Machine Intelligence 4 D Michie and B Meltzer eds 1968

6 C GREEN
The application of theorem proving to question-answering systems
Stanford Research Institute CS-138 June 1969

7 C H KELLOGG
A natural language compiler for on-line data management
Proceedings AFIPS 1968 pp 473-492

8 R A KIRSCH
The computer interpretation of English text and picture patterns
IEEE Transactions on Electronic Computers EC-13 1964 pp 363-376

9 M KOCHEN
Automatic question-answering of English-like questions about simple diagrams
Journal of the ACM Vol 16 No 1 January 1969 pp 26-48

10 M MINSKY ed
Semantic information processing
The MIT Press Cambridge Massachusetts 1968

11 A NEWELL H A SIMON
GPS, a program that stimulates human thought
In Computers and Thought E Feigenbaum and J Feldman eds McGraw-Hill New York 1963

12 R J ORGASS
Logic and question answering
IBM Thomas J Watson Research Center RC 3122 #14585 December 16 1970

13 H E POPLE JR
A goal-oriented language for the computer
Reprinted in Representation and Meaning Experimenting with Information Processing Systems Simon & Siklossy eds Prentice-Hall Englewood Cliffs NJ 1972

14 M R QUILLIAN
Semantic memory
PhD dissertation Carnegie-Mellon University 1966 222pp

15 M R QUILLIAN
The teachable language comprehender: A simulation program and theory of language
Communications of the ACM Vol 12 No 8 August 1969 pp 459-476

16 E J SANDEWALL
Representing natural-language information in predicate calculus
Artificial Intelligence Project Memo AIM-128 Stanford July 1970

17 R W SCHWARCZ J F BURGER R F SIMMONS
A deductive question answerer for natural language inference
SDC Report—SP-3272 November 15 1968

18 R M SCHWARCZ
Towards a computational formalization of natural language semantics
Presented at the International Conference on Computational Linguistics in Sanga-Saby-Kursgaard Sweden September 1969

19 R C SHANK L G TESLER
A conceptual Parser for natural language
Proceedings of the Joint International Conference on Artificial Intelligence 1969 pp 569-578

20 R C SHANK
Finding the conceptual content and intention in an utterance in natural language conversation
Second International Joint Conference on Artificial Intelligence September 1971 pp 444-454

21 S C SHAPIRO
A net structure for semantic information storage, deduction and retrieval
Second International Joint Conference on Artificial Intelligence September 1971 pp 512-523

22 R F SIMMONS
Answering English questions by computer: A survey
Communications of the ACM Vol 8 No 1 January 1965 pp 53-70

23 R F SIMMONS
Natural language question—Answering systems
Communications of the ACM Vol 13 No 1 January 1970 pp 15-30

24 F B THOMPSON
English for the computer
Proceedings AFIPS FJCC Vol 29 pp 349-356

25 W A WOODS
A procedural semantics for a question—Answering machine
Proceedings AFIPS FJCC 1968 pp 457-471

An information processing approach to theory formation in biomedical research*

by H. POPLE and G. WERNER

University of Pittsburgh
Pittsburgh, Pennsylvania

INTRODUCTION

The extensive literature on modeling of biological systems published in the past decade reflects the growing expectation that theories of biological functions can be subject to more exacting tests of consistency with the natural system, and yield more powerful predictions if embodied in the formal structure of computer programs.

One principal aspect of the usefulness of such models of biological systems is attributable to their capability to generate predictions under conditions which transcend the human mind's limited capacity to trace implications through long chains of causal connections, notably when there exist possibilities for multiple interactions between components of a system designed for insuring some degree of homeostasis. Moreover, once sufficiently elaborate, refined and acceptable to a community of investigators, a model could be expected to generate upon request observable facts at various levels of resolution and generality, without having all of these facts explicitly encoded and stored. In this form, the model could be regarded as an information storage and retrieval device from which data can be generated by suitable programs and algorithms.

Irrespective of the particular purpose a model may subserve, it can be considered as an aggregate of components whose initial properties are given in the form of a set of definitions, and whose interactions are determined by suitable transfer functions. The modeler's task is, therefore, twofold: in the first place, suitable algorithms need to be selected to represent the input-output transfer functions of the individual model components. Second, the task consists in designing an explicit structural arrangement of the components, and a pattern of information flow between them, which is suggested or implied by the information in the data base, and which enables the model to mimic the performance of the natural system.

In some sense, one may anticipate that the potential usefulness of computer based models grows with the complexity of the natural domain they represent. However, as the latter increases, model building itself turns into an increasingly more formidable task, frequently complicated by either incompleteness or ambiguity of some of the available observational data. The implication of this is that the relation between a theory and its data base on the one hand, and a corresponding model on the other hand, need not be unique; instead, each given set of observational data of a certain complexity can generate a class of models. Consequently, in order to work toward a parsimonious model compatible with the available evidence, the modeler must engage in the iterative process of identifying alternative interpretations of the data, selecting between them, and evaluating their consequences. In those natural domains whose complexity makes modeling most desirable, this information processing task tends to assume a degree of complexity that impedes model development. To explore ways of assisting the modeler in this task, we have inquired into the nature of the information transactions underlying the modeling of complex biological systems. In the following sections, we discuss the nature of this information processing task. We, then, focus on the problem of *structure optimization* of neural networks, that is: the adaptive process of fitting together a neural network which, when simulated, gives rise to behavior analogous to that observed in the natural domain. Finally, we discuss computational procedures designed to aid in this process.

SOME ASPECTS OF THE MODELING TASK

In our work with models of complex neural control systems, we have come to recognize the need for drawing

* This work was supported by NIH Contract 70-4121.

a sharp distinction between the concept of "hypothesis model" and that of "simulation model." The hypothesis model is what the investigator carries in his head; it consists of a collection of facts and a set of interpretive rules that allows the researcher to perceive relationships in the data, make predictions concerning consequences of untried experiments, etc. A simulation model, on the other hand, is a computer program in which the researcher can express the essential features of his internal hypothesis model, enabling rigorous tests of validity. As such, it requires a degree of explicitness which goes beyond that commonly available in the hypothesis model. We emphasize this dual aspect of the modeling process in order to bring into focus the interplay that takes place between data, hypothesis, and simulation.

In many cases, the implications of biological data, in terms of the scientist's hypothesis models, may give rise to a multitude of possible interpretations. For example, the finding that "electrical stimulation of Deiters Nucleus elicits IPSPS (inhibitory postsynaptic potentials) in gamma-motorneurons" could suggest a variety of structural connectivities; some of those are:

(a) an inhibitory pathway between the cells of Deiter's nucleus and the flexor gamma-1-motorneurons;

(b) an excitatory pathway between the cells of Deiter's nucleus and some spinal-interneurons which, in turn, inhibit gamma-motorneurons on the extensor side;

(c) orthodromic and antidromic conduction along collaterals of axons which originate in some cell group with inhibitory connections to, both, Deiter's nucleus and certain gamma-motorneurons.

While it may be possible to rule out certain of these alternatives on the basis of other data, the researcher is invariably forced—in the course of constructing a simulation model—to make essentially arbitrary choices for which there is no conclusive evidence, either in the theory or in the data.

If a hypothesis model is subjected to a simulation test that is successful in the sense that the latter exhibits behavior which is consistent with the predictions by the former, then the researcher is reinforced in the choices that were made. On the other hand, if this correspondence is not attained, a reexamination of data, of hypothesis model, and of simulation model performance may direct the investigator to more rational choices. This is one of the principal benefits of using simulation: it can reduce the degrees of freedom available to the model builder, and can force a restructuring of his hypothesis model which, in the absence of a simulation experiment, might not take place.

However, in order to exploit the full value of negative simulation results, the researcher needs to know much more than simply that the results were unsatisfactory. Of course if that is all he knows, he can still proceed with revision of the model and additional experimentation. He might, for example, go to some point in his hypothesis model where an arbitrary choice from among alternative interpretations of the data had been made, and simply try another choice. In following such an exhaustive search procedure, he may be forced to go through a large number of variations of the simulation model until one is found that supports his theoretical model. Provided he has the patience to persevere in this endeavor, it could lead ultimately to an enhancement of his theory.

As is the case in most such search processes, it is reasonable to expect that some heuristic procedure which makes use of the detailed information developed in the course of a simulation, would provide substantial improvement in terms of the time and effort required to obtain a fit with the data. The important questions are:

(a) what kind of information does the researcher need; and

(b) in what form will this information be of most use to him.

In a later part of this paper, we will describe the kinds of information support systems that we currently have under development to provide relevant inputs for the restructuring of hypothesis models. Their development was engendered by the experience gained in the design of a simulation model of the central neural control of skeletal muscle tone and reflexes. As a prelude to the discussion that follows, it will be useful to describe at this point the way in which one observes the macro- and micro-behavior in this motor control simulator, further details of which are described in the appendix.

OBSERVATION OF MODEL BEHAVIOR

The main tool for observing macro-behavior in the motor-control system models is the ANALYZE program (see appendix), which is used to develop behavioral comparisons of two states of a model (one of which typically represents the normal condition, the other some pathology). This program provides a verbal output that describes deviations in behavior in one state of the model, relative to another, along certain key dimensions. Changes in such attributes as amplitude and rate of muscle contraction, force required for smooth passive stretch, frequency of oscillatory behavior, and others are perceived and reported to the user by this processor.

While in the model-building/data-fitting phase of his work, the researcher can use ANALYZE to test his developing hypotheses concerning representation of pathology in the model. For example, he might know from a review of the data that a lesion affecting the motor-cortex (area 4) would be expected to yield predominance of extensor muscle tone in the organism. Moreover, he might already have established in his model a number of connections emanating from the motor-cortex that could explain such a change in behavior. There might, for instance, be an excitatory pathway from this cortex to alpha motoneurons supplying flexor muscles. In approaching a simulation of this pathology, the researcher would therefore start with a number of pre-conceptions and expectations. In particular, since he intends that the structures which appear in the model should have a functional correspondence with their counterparts in the real system, he would expect (and indeed would insist for the model to be considered acceptable) that destruction of the motor-cortex of his model would be comparable, in its behavior manifestations, to ablation of the real structure in the living organism.

Unfortunately, it often happens in the process of model building that such pre-conceptions are not supported by the simulation results, and it becomes then necessary for the researcher to engage in a kind of post-mortem where he reviews his assumptions and the simulation results in an attempt to assess what went wrong. Any number of things could have happened. For example, the researcher might have expected some pathway to be 'turned off' in the pathology because its source of excitation would be all or partially removed. He may find on closer examination that this particular pathway is not sufficiently active in the normal model; this would of course explain the failure to induce the anticipated pathology. There may also be other, more complicated reasons for the model to deviate from the anticipated behavior: for instance the simulated lesion may give rise to widely dispersed changes in the activity pattern that are difficult to trace in the hypothesis model, and which are therefore less likely to have been anticipated during the theoretical analysis.

The ANALYZE program is of limited value in identifying such failures in the reasoning process. While it provides an answer to the question: "How did the model fail?" it does not address the equally important question "Why?" In order to gain insight into *why* a model fails, we must examine the micro-structure of model behavior and the information flow at key structures along various pathways. The analysis of this information flow in the model typically requires an extensive series of simulation tests, involving various diagnostic procedures in both the normal state of the model and in the intended pathology. Once this analysis has been completed, the researcher must then determine what these new insights concerning information flow imply with regard to his hypothesis models, make the appropriate adjustments, and try again.

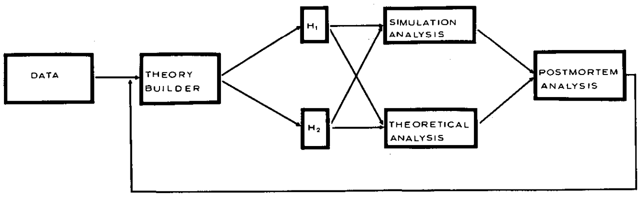


Figure 1

The basic tasks involved in this modeling process are illustrated by the flowchart of Figure 1. Here, the nodes H1 and H2 represent hypothesis models corresponding to the two experimental conditions being compared. The arcs connecting these nodes to the box labeled 'Simulation Analysis' correspond to sequences of commands of the form: 'Cut,' 'Tie,' 'Block,' 'Facilitate'—operators which have been provided as part of the motor-control simulator package for use in constructing various versions of the simulation model (see appendix).

Other aspects of the flow chart are less well defined, being part of the informal analysis engaged in by the investigator. The box labeled 'Theoretical Analysis,' for example, represents the researcher's attempts to rationalize the expected change in macro-behavior on the basis of probable changes in micro-behavior; that is, it reflects his attempts at predicting the cause and effect relationships that are operative in the pathology under consideration.

These predictions feed into a comparator labeled 'Post-Mortem Analysis,' which also receives the actual observations that result from running the simulation model. In the event that prediction and observation fail to match, an 'error signal' is generated and is fed back to the box labeled 'Theory Builder' causing a restructuring of the hypothesis models and a reiteration of the verification cycle.

Whenever this procedure requires an alteration of the hypothesis model, the experimenter must examine whether all experimental conditions accounted for in the previous version of the hypothesis model, are still satisfied. Thus, the flowchart of Figure 1 describes just one phase of the total information processing task associated with the development of a general hypothesis model that accounts for a number of different experimental results.

INFORMATION SUPPORT SYSTEMS

The modeling task described in the preceding sections comprises an extremely complex problem in information processing. Although much of the work seems to be of a routine and mechanical nature—tedious is perhaps the right word—there has been no methodology available to relieve the inordinate burden this procedure places on the researcher. To provide an effective research tool, capable of supporting a wider spectrum of the investigator's activity rather than merely the mechanics of simulation, there is need for computational procedures that will aid in the theory building phases of the total information process, and also facilitate theoretical and post-mortem analysis.

In order to state these objectives more precisely, let us consider again the dilemma of a researcher who is attempting to develop theoretical models to explain data pertaining to the motor-control system. He has access to a number of different kinds of experimental results that can be grouped according to the following classifications:

(a) anatomical
(b) physiological
(c) pathological
(d) pharmacological

Assuming fixed algorithms for the neural elements and the peripheral motor and sensory components of the system, the researcher's task is that of developing a theoretical model—in the form of a neural connectivity network—which is consistent with the observational data of each of these four domains, and the implications thereof.

What we have set out to do is reduce this task to a computational procedure. Toward this end, we have designed and implemented an inferential processor that can be used to analyze data and propose hypothesis models. Because of its central role in the total information system concept, we discuss next, in some detail, the conceptual basis of this theory building program.

THE THEORY BUILDER

The first step in developing computational procedures for dealing with theory is a formalization of that theory. There are, of course, a number of computational procedures extant—e.g., resolution, model elimination—for dealing with the deductive aspects of theory.[1,2] However, our problem is concerned not so much with what the theory implies as with what theory is implied. This converse problem, referred to variously in the literature as 'apagoge' or 'abduction,'[3] is put succinctly by McCulloch:[4]

> ". . . *abduction* starts from the rule and guesses that the fact is a case under that rule: All people with tuberculosis have bumps; Mr. Jones has bumps; perhaps Mr. Jones has tuberculosis. This, sometimes mistakenly called an 'inverse probability,' is never certain but is, in medicine, called a diagnosis or, when many rules are considered a differential diagnosis, but it is usually fixed, not by a statistic, but by finding some other observable sign to clinch the answer."

If we express the syllogism contained in this argument symbolically:

$$\begin{array}{r} Tx\supset Bx \\ Ba \\ \hline \text{Perhaps}\quad Ta \end{array}$$

it is clear that a stronger conclusion 'Ta' would be fallacious, and the qualifier 'perhaps' is a necessary part of the abduction. The uncertainty that attaches to such a result can be lessened, however—as McCulloch points out—by finding additional evidence to clinch the case. Even another abduction argument can lend support; for instance, if we also have:

$$\begin{array}{r} Tx\supset Cx \\ Ca \\ \hline \text{Perhaps}\quad Ta \end{array} \qquad \text{and} \qquad \begin{array}{r} Px\supset Cx \\ Ca \\ \hline \text{Perhaps}\quad Pa \end{array}$$

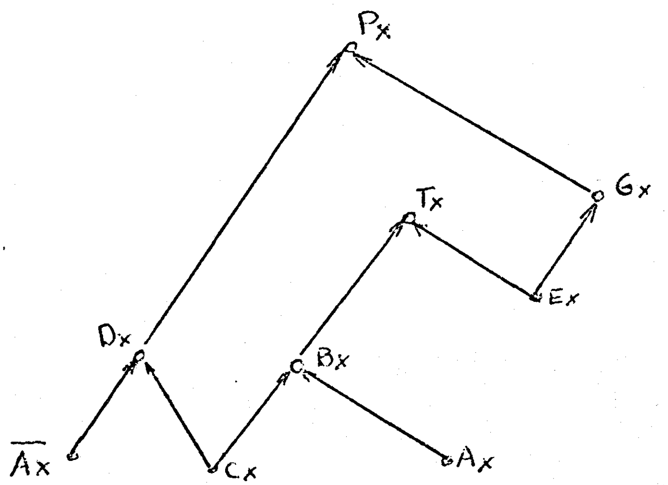
the occurrence of two distinct data supporting 'Ta,' while only one supports 'Pa,' would cause most of us to hypothesize 'Ta.'

Such support can also come indirectly through a somewhat lengthier chain of reasoning. Consider for example the set of rules:

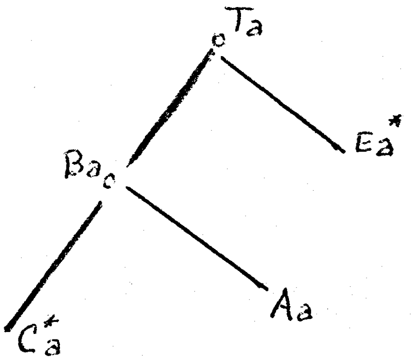
$Tx \supset Ex \quad Tx \supset Bx \quad Bx \supset Cx \quad Px \supset Gx \quad Bx \supset Ax$

$Dx \supset Cx \quad Gx \supset Ex \quad Px \supset Dx \quad Dx \supset \overline{Ax}$

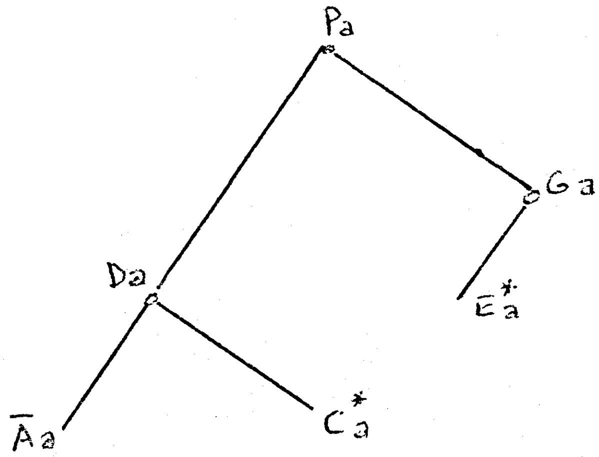
We can illustrate the network of implications contained in these premises graphically as follows, where the upward branches at any node denote 'reverse implication':



If we observe the data: {Ca, Ea}—then we might reasonably hypothesize: {Ta, Ba, Aa}—since this collection of nodes defines a sub-graph of the net that is mutually supportive:



Since only 2 of these 5 nodes have actually been observed, however, any assumption concerning the others is merely theory. An alternative theory that might be put forth to account for these same data would be:



where again, only 'Ca' and 'Ea' correspond to the actual observations.

If we have the opportunity to ask questions, we often are able to discriminate between alternative theories. For example, in this case if we ask "Aa?" then one or the other of the proposed theories will necessarily be set aside. Note that although the pair $\{A \supset B, B\}$ does not yield a definite conclusion, the pair $\{A \supset B, \bar{B}\}$ does.

Provided we have data that can be structured in the form of an implicative or causal net—as above—we can clearly write heuristic computer programs that engage in the abductive process of theory building. The objective of such a program would be to find a single connected subgraph that 'covers' all of the observations; or, failing that, a pair of such subgraphs; or a triple, etc. Since each additional subgraph included in a theory contains an additional basic assumption (top node), the goal of parsimony in theory development would provide a rule for selection from among alternative hypotheses even in the absence of other criteria. If, in addition, we have the ability to ask questions of the user, of a simulation model, and of the structured and unstructured data files, the power of our program to develop sound theory is clearly enhanced.

We have encoded and successfully demonstrated a heuristic processor, using a combination of LISP[10,11] and GOL[5] routines, that constructs hypothesis models on the basis of the heuristic procedure outlined above. We are in the process of interfacing this Theory Building program with a natural language query system that also possesses some inferential capability, subsequently referred to as ENGOLISH.[6] This makes it possible that:

(a) an uninterpreted literature data base, in the form of ENGOLISH data structures, can be accessed by the theory builder program, and

(b) the theory builder program can be invoked to develop hypothesis models while engaged in an ENGOLISH dialog with the user; thus avoiding inconsistencies, ambiguities, and incompleteness in the data base recorded.

The set of rules that generates the abduction net of the theory builder is encoded in a table of associations called SUGGESTS, the present form of which is as follows:

```
(GOLDEF  SUGGESTS
 (S0000
  (EXT  ((P Q R S U V W X Y Z))
        ((PROJECTS P Q)  ((POLARITY X)  (CELL-TYPE P Y)  (CELL-TYPE Q Z))
                         ((PATHWAY MONOSYNAPTIC X Y Z)))
        ((ELICITS (P Q) (R S))
         ((PATHTYPE P R W X))
         ((PATHWAY W X Q S)))
        ((PATHWAY POLYSYNAPTIC P Q R)
         ((MATCHUP P Y W)  (DISTINCT Q S)
                           (DISTINCT S R)
                           (ORTHODROMIC Z))
         ((PATHWAY MONOSYNAPTIC Y Q S)  (PATHWAY Z W S R)))
        ((PATHWAY ANTIDROMIC P Q R)
         ((ORTHODROMIC X)  (MATCHA P X Y W)
                           (DISTINCT Q S)
                           (DISTINCT R S))
         ((PATHWAY MONOSYNAPTIC Y S Q)  (PATHWAY X W S R)))
        ((INTERRUPTED  (PATHWAY MONOSYNAPTIC P Q R) U)
         ((GOLDIG ((LAMBDA NIL (OR (DESTROYED R)  (DESTROYED Q)))) U))
         ((PATHWAY MONOSYNAPTIC P Q R)))
        ((INTERRUPTED (PATHWAY POLYSYNAPTIC P Q R) U)
         ((MATCHUP P Y W)
          (DISTINCT Q S)
          (DISTINCT S R)
          (ORTHODROMIC X)
          (GOLDIG ((LAMBDA NIL (OR (DESTROYED Q)  DESTROYED S)))) U))
         ((PATHWAY MONOSYNAPTIC Y Q S)  (PATHWAY X W S R)))
        ((INTERRUPTED (PATHWAY POLYSYNAPTIC P Q R) U)
         ((MATCHUP P Y W)  (DISTINCT Q S)
                           (DISTINCT S R)
                           (ORTHODROMIC X))
         ((PATHWAY MONOSYNAPTIC Y Q S)
          (INTERRUPTED (PATHWAY X W S R) U)))
        ((INTERRUPTED (PATHWAY ANTIDROMIC P Q R) U)
         ((ORTHODROMIC X)
          (MATCHA P X Y W)
          (DISTINCT Q S)
          (DISTINCT R S)
          (GOLDIG ((LAMBDA NIL (OR (DESTROYED Q)  (DESTROYED S)))) U))
         ((PATHWAY MONOSYNAPTIC Y S Q)  (PATHWAY X W S R)))
        ((INTERRUPTED (PATHWAY ANTIDROMIC P Q R) U)
         ((ORTHODROMIC X)  (MATCHA P X Y W)
                           (DISTINCT Q S)
                           (DISTINCT R S))
         ((PATHWAY MONOSYNAPTIC Y S Q)
          (INTERRUPTED (PATHWAY X W S R) U)))))
```

This table is encoded as a GOL extensional data structure[5] in which the symbols: (P, Q, R, S, U, V, W, X, Y, Z) stand for universally quantified variables. Each entry of the table consists of three parts, (A B C), which can be read:

"C implies A under conditions B"

For example, the first entry would be interpreted:

> A(P, Q, X, Y, Z) pathway (monosynaptic, X, Y, Z) $\supset$ projects (P, Q); *where* (polarity (X) $\wedge$ cell-type (P, Y) $\wedge$ cell-type (Q, Z).

In this expression,

- 'pathway' predicates a monosynaptic connection, of polarity X between cells of type Y and type Z
- 'projects' predicates a fiber projection between neural structures P and Q.
- 'polarity' is the property: {excitatory, inhibitory}; and
- 'cell-type' is a relation that associates the names of neural structures with the various neural populations subsumed thereunder.

The operation of abduction, using the SUGGESTS data structure entails an associative access whereby some observational datum (e.g., 'projects motor-cortex-4 flexor-alphas') is matched against the first position (A) of appropriate entries of the table, with the result a hypothesis (C) qualified by the expression (B). The reason for employing quantifiers and qualifiers in the SUGGEST-Table is that this permits large numbers of nodes to be represented compactly by a single entry; furthermore, it enables the use of much more powerful search techniques than would otherwise be possible. This refined principle of operation would appear to have the same relation to simple abduction that resolution holds to Herbrand expansion.[2]

While these concepts and their relationship to other paradigms of artificial intelligence will be the subject of subsequent publications,[7] we present at this juncture an illustration of their operational aspects. For this purpose, we consider an application of the theory builder in the context of a collection of typical neurophysiological data giving rise to alternative hypothesis models.

In a discussion of the diversity of the effects of cerebellar stimulation, Eccles et al.[8] describe the various pathways which mediate the inhibitory influence of the cerebellar cortex on a cell group in the vestibular nuclei. Some aspects of this discussion can be stated in terms of the following propositions:

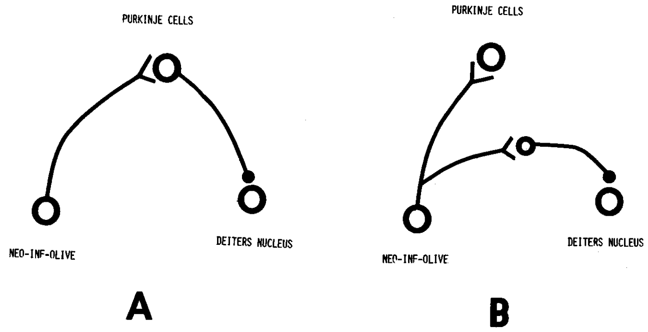


Figure 2—Schematic diagrams of the neuronal connectivity proposed by the theory building program. The empty circles are neural structures whose names correspond to those appearing in the program dialogue. The connecting pathways ending with an open branch signify excitatory, those ending with a solid circle inhibitory connections

(1) the inferior olive projects monosynaptically *via* the climbing fibers to the cerebellar cortex;
(2) activity engendered by electrical stimulation in the main inferior olivary nucleus (neo-olive) elicits EPSP's in Purkinje cells;
(3) activity engendered by electrical stimulation in the main inferior olivary nucleus elicits IPSP's in Deiters Nucleus.
(4) electrical stimulation of Purkinje cells elicits IPSP's in Deiters Nucleus.

To deal with these propositions meaningfully, some general background knowledge is required which finds its representation in the data encoded in the SUGGEST-table, given previously:

(a) fiber projections identified by neuro-anatomical procedures *are* monosynaptic pathways;
(b) electrical stimulation in the typical experimental procedure elicits activity not only in the target cells, but also in fibers and their termination in the vicinity of these cells.
(c) impulse traffic in fiber tracts under condition of artificial electrical stimulation can be either ortho- or antidromic, or both.
(d) chronic lesion of a structure interrupts impulse transmission (both, ortho- and antidromic) in all pathways of which this structure is a part.

As illustrated by the following dialogue and in Figure 2a, when supplied with just these propositions and facts, the theory builder proposes a partial model, consisting of one excitatory and one inhibitory monosynaptic pathway:

```
*(STRUCTURE  INFERIOR-OLIVE  NEO-INF-OLIVE  PALEO-INF-OLIVE)
NIL
*(STRUCTURE             CEREBELLAR-CORTEX           PURKINJE-CELLS
                       GOLGI-CELLS          BASKET-CELLS)
NIL
*(STRUCTURE             VESTIBULAR-NUCLEI           DEITERS-NUCLEUS
                        MEDIAL-VN           SUPERIOR-VN      DESCENDING-VN)
NIL
*(OBSERVATION           (PROJECTS           INFERIOR-OLIVE
                        CEREBELLAR-CORTEX))
NIL
*(OBSERVATION           (ELICITS            (STIM     NEO-INF-OLIVE)
                                            (EPSP     PURKINJE-CELLS)))
*NIL
*(OBSERVATION           (ELICITS            (STIM     NEO-INF-OLIVE)
                                            (IPSP     DEITERS-NUCLEUS)))
NIL
*(OBSERVATION           (ELICITS            (E-STIM   PURKINJE-CELLS)
                                            (IPSP     DEITERS-NUCLEUS)))
NIL
*(PONDER)
(PROPOSE MODEL:
     ((PATHWAY          MONOSYNAPTIC        EXCITATORY
                        NEO-INF-OLIVE       PURKINJE-CELLS)
      (PATHWAY          MONOSYNAPTIC        INHIBITORY
                        PURKINJE-CELLS      DEITERS-NUCLEUS)))
```

If now provided with the additional observational datum that a chronic lesion of the inferior olive abolishes the inhibitory response in Deiters nucleus that follows electrical stimulation of Purkinje cells, the program recognizes the inconsistency between the implications of this finding and its internal model. This gives rise to a restructured hypothesis model, as follows (see Figure 2 B):

```
*(EXPLAIN          (ABOLISHES          (CHRONIC-LESION          INFERIOR-OLIVE)
                                       (ELICITS  (E-STIM  PURKINJE-CELLS)
                                                 (IPSP  DEITERS-NUCLEUS)
                                                                   )))
(THIS FINDING INCONSISTENT WITH PREVIOUS MODEL)

(PROPOSE PARTIAL MODEL):

          ((PATHWAY     MONOSYNAPTIC       EXCITATORY
                        NEO-INF-OLIVE      PURKINJE-CELLS)
           (PATHWAY     POLYSYNAPTIC       INHIBITORY
                        NEO-INF-OLIVE      DEITERS-NUCLEUS)))
```

The operation of the theory building program starts with the definition of the domain of discourse: this is accomplished by use of the STRUCTURE—command. The first argument of this command designates the name of a neural structure; subsequent arguments identify its components. The command OBSERVATION is used to perform a preliminary encoding of an observational datum which may be either a statement concerning automical connections, or a description of functional relations. Supplied with these facts, PONDER is called to invoke the abduction process which generates a parsimonious hypothesis model that accounts for the data supplied thus far.

The continuation of the model building process consists of the addition of new data and their evaluation relative to the existing model. The macro command EXPLAIN is available for this purpose: in the first place, it enables the user to supply a new observational datum; the program, then, ascertains whether this new fact is consistent with the existing hypothesis model: if this is the case, it renders an account of that part of the existing model which has been enriched, either by way of confirmation of an existing aspect of the model, or by expanding its domain. Alternatively, the program may discover an inconsistency between the model it contains and the new datum, in which case it reverts to PONDER to attempt the generation of a new model which accommodates all of the facts it has been given up to that time.

ON MECHANIZED MODEL BUILDING—AN OUTLOOK

Having presented and illustrated the mode of operation and capability of the theory building program, we are now in a position to assess its role within the total information support system for model building. With reference to the flow chart of Figure 1 which dissected the process of model building into identifiable steps and the interrelations between them, we have demonstrated that the theory building program can generate hypotheses on the connectivity of model components which correspond to the boxes labeled H_1 and H_2 in that Figure: that is, a connectivity that may obtain under normal, and also satisfies some abnormal, conditions; the latter consists in the example given of an artificial, experimental lesion. The hypotheses concerning connectivity are of a form which permits them directly to be represented in the simulation model by means of the CUT and TIE commands (see appendix).

The example illustrates that the formation of hypothesis models can be mechanized, which was the principal objective we have set out to accomplish. Beyond this, the theory building program contains features which enable it to contribute also in other ways to the development of theoretical models; namely, by posing questions, and by proposing experiments to be conducted in the "simulation laboratory." To this end, the EXPLAIN command can be used to generate predictions and expectations for meaningful post-mortem analysis (see Figure 1): it directs the investigator's attention to those components and pathways of the simulation model which are expected to be most prominently involved in the anticipated behavioral differences. The outcomes of the theoretical and the simulation analysis can now be compared; if significant differences obtain—which would signify the negation of some implication of the hypothesis model—this new datum can be fed back and employed in the restructuring of the hypothesis.

Operating in this manner, the theory-building program can now be envisaged to perform the structure optimization of the neural network, relieving the investigator from the tedium of the iterative process of fitting observational data into a useful simulation model. Instead, once he has defined the first principles which will serve as the basis of theory formation—in the form of rules in the SUGGESTS data structure—he is able to assess continuously the sufficiency and necessity of his concept of the natural domain under study, and the consistency of this concept with newly acquired data.

APPENDIX

Modeling tools of the motor control system simulator

The motor control system simulator is a processor that can be used to construct and evaluate simulation models of neural control mechanisms which play a role in the regulation of certain skeletal muscle reflexes and of muscle tone. This requires, in the first place, that the investigator have available a versatile set of commands to define neural structures and to interconnect them by pathways of suitably chosen length (i.e., conduction time). These commands are intended to enable the investigator to formalize his conception of the system under study in the form of a model that can be altered conveniently in accord with new evidence; that can reflect some experimentally induced condition; and that can mimic disease states as well as effects due to administration of pharmacological agents. The second requirement was to provide programming tools which would exercise the model in a manner analogous to the tests used by the experimentor to ascertain the functional capabilities of the natural system. Thirdly, it seemed to us necessary to provide the ability for examining the performance of components in the model, and of the model as a whole, at different levels of resolution, comparable to the procedures used by the experimenter: at one time he may be interested in the fine temporal pattern of activity in one single node of the model; and another time, in a time average of the activity, or in the trend; and at some other occasion, in the gross overt performance of an effector organ (in the case of the prototype model: muscle length and tone), or merely in the deviation of this performance from a normal baseline; finally, the investigator may wish to recognize the relation between performance of an effector organ and the pattern or activity flow between the interacting components of the (neural) control system. Accordingly, programs to extract and perceive any one of these aspects of the model's behavior are available for selection by the user.

Algorithms of system components

The basic element of the simulated neural system is a stylized neuron whose activity state is described by a quadruple of numbers, each ranging over the numerical values from zero to four. These numbers designate, in this sequence, output, threshold, input, and a term representing a "driving force." The latter is a constant which sums algebraically with the value of the threshold and enables the user to set a positive or negative bias value on the neuron computation.

These formal neurons can be connected by excitatory or inhibitory links to form a finite state machine in which each neuron can receive an arbitrary number of inputs. These are summed algebraically to determine the net input. Computation within the neural network is deterministic and proceeds in discrete time steps, each corresponding to the synaptic delay of approximately 1 millisecond duration in real time. At each time slice, the input and output values of each neuron, representing a neurophysiologically characterized neuron population, are updated on the basis of the following algorithms:

```
if $((\text{input}(t-1) > \text{thresh})$ and
  $(\text{output}(t-1) < \text{input}(t-1))$ and $(\text{thresh} \neq 4))$:
    $\text{output}(t) = \text{output}(t-1) + 1$
if $((\text{thresh} = 4)$ or $(\text{input}(t-1) < \text{thresh})$ or
  $(\text{output}(t-1) > \text{input}(t-1))$:
    $\text{output}(t) = \text{output}(t-1) - 1$
else: $\text{output}(t) = \text{output}(t-1)$
$\text{input}_j(t) = \sum_{k \epsilon A} (\text{output}_k(t - d_{kj}))$
```

(where A is the set of structures K that are connected, *via* delays of lengths d_{kj}, to the j_{th} neuron.

TABLE I—Periphery Algorithms

1. motor-output(t) := flexor alpha-motorneuron
2. ext-motor-output(t) := extensor-alpha-motorneuron
3. gamma1-output(t) := flexor-gamma1-motorneuron
4. gamma2-output(t) := flexor-gamma2-motorneuron
5. ext-gamma1-output(t) := extensor-gamma1-motorneuron
6. ext-gamma2-output(t) := extensor-gamma2-motorneuron
7. flex-length(t+1) := flex-length(t) +
   flex-mult * (golgi-output(t) − motor-output(t))
8. ext-length (t+1) := ext-length(t) +
   ext-mult * (ext-golgi-output(t) − ext-motor-output(t))
9. golgi-output (t+1) := (ext-length (t+1) /
   (ext-length (t+1) + flex-length (t+1)))*
   (ext-motor-output(t) + motor-output(t))
10. ext-golgi-output (t+1) := (flex-length (t+1)/
    (ext-length (t+1) + flex-length (t+1)))*
    (ext-motor-output(t) + motor-output(t))
11. chain-output (t+1) := scale (flex-length (t+1) −
    chain-length(t), kchn)
12. chain-length (t+1) := chain-length(t) +
    cmult * (chain-output (t+1) − gamma2-output(t));
    (clip at zero)
13. ext-chain-output (t+1) := scale (ext-length (t+1) −
    ext-chain-length(t), kchn)
14. ext-chain-length (t+1) := ext-chain-length(t) + ecmult *
    (ext-chain-output (t+1) − ext-gamma2-output(t));
    (clip at zero)
15. ext-bag-output (t+1) := scale (ext-length (t+1) −
    ext-bag-length(t), kbag)
16. ext-bag-length (t+1) := ext-bag-length(t) + ebmult *
    (ext-bag-output (t+1) − ext-gamma 1-output(t));
    (clip at zero)
17. ext-bag-rate := ext-mult (ext-golgi-output(t) −
    ext-motor-output(t)) − ebmult* (ext-bag-output(t+1) −
    ext-gamma 1-output(t)) + ext-bag-output (t+1)
18. bag-output (t+1) := scale (flex-length (t+1) −
    bag-length(t), kbag)
19. bag-length (t+1) := bag-length(t) + bmult*
    (bag-output (t+1) − gamma 1-output(t));
    (clip at zero)
20. bag-rate := flex-mult (golgi-output(t) − motor-output(t)) −
    bmult* (bag-output (t+1) − gamma 1-output(t)) +
    bag-output (t+1)
21. golgi-tendon (t+1) := golgi-output (t+1)
22. ext-golgi-tendon (t+1) := ext-golgi-output (t+1)
23. nuclear-chain (t+1) := chain-output (t+1)
24. nuclear-bag (t+1) := bag-rate (t+1)
25. ext-nuclear-chain (t+1) := ext-chain-output (t+1)
26. ext-nuclear-bag (t+1) := ext-bag-rate (t+1)

The neural network controls the length of reciprocally connected flexor and extensor muscles through the algorithm of *table* 1 which also computes for each value of muscle length and gamma motorneuron output the feedback signals generated by muscle spindles (nuclear bag and chain organs) and Golgi tendon stretch receptors as schematically illustrated in Figure 3 (for review, see Reference 9). The connecting links between the neural control network and the muscle effector system are provided by three motor neurons, each, on flexor and extensor side; and by afferent pathways originating from the length and tension transducers of muscles and tendons. Items 1 through 6 of Table 1 designate the relationship between the respective motor neurons and the variables appearing in the periphery algorithms; items 21 through 26 designate the sources of feed back signals. The parameters appearing in these algorithms have the following meanings:

flex-mult [ext-mult]: scale factor used to translate the tension changes of flexor [extensor] extrafusal muscle fibers into length changes;

cmult [ecmult]: a scale factor to accomplish the same for intrafusal muscle fibers of the nuclear chain receptors;

bmult [ebmult]: a scale factor for intrafusal muscle fibers of the nuclear bag receptors;

scale: a scaling function that uses the tabular functions KBAG and KCHN (see later) to translate length differences between extra- and intrafusal muscle fibers of nuclear bag and nuclear chain receptors, respectively, into the static component of the transducer outputs.

It might be useful at this point to comment on issues of implementation. We have had access for this development to a medium scale PDP-10 system, on which a superb LISP interpreter[10,11] is available. To provide the user a convenient means for interacting with his simulation models, we have encoded a set of supervisor and exercisor routines—to be described presently—that operate in the context of this LISP interpreter. Since LISP is singularly inefficient in its provision for numerical computation, those algorithms described above which define computations at the neural nodes as well as in the peripheral effector and sensor organs have been encoded in FORTRAN, and the resulting compiled modules have been incorporated in the LISP system as callable subroutines. Thus in the discussion that follows, although the notation used to illustrate features of the simulation system if that of LISP, the reader should be aware that the actual simulation results are produced by execution of the appropriate FORTRAN subroutines, upon direction by the LISP supervisor programs.

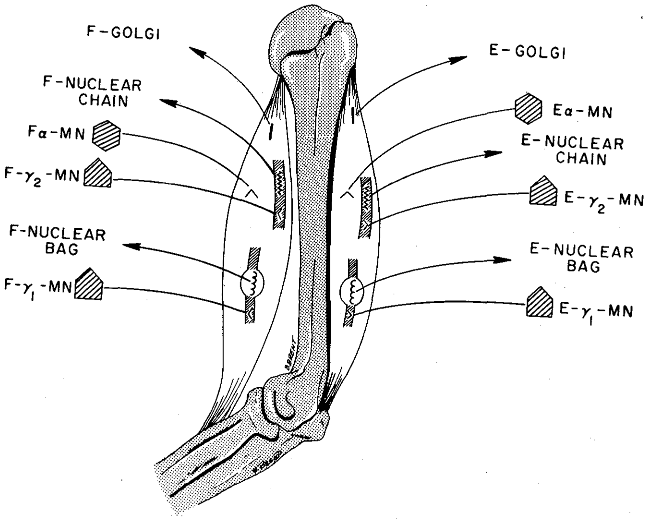


Figure 3—Schematic display of an antagonistic muscle pair (F = flexor, E = extensor muscle), and their efferent and afferent innervation. α — MN = motorneurons supplying the extrafusal muscle fibers; γ_1- and γ_2-MN's: gamma-motorneurons supplying the muscle fibers of nuclear bag and nuclear chain transducers, respectively, from which the afferent nerve impulses signaling muscle length originate. Golgi = receptors situated in muscle tendons and signaling muscle tension

Command structure for model building

The devices needed to build a model and to alter connections, add or delete structures, specify delays (i.e., conduction times between neural structures, in milliseconds real time) and to pass numerical parameters to the computational algorithm are available in the form of a set of LISP functions. Examples of some steps in the building of a model are illustrated in the following interactive dialogue with the model building supervisor:

```
*(ADD NEURON @N-RUBER 1$
N-RUBER
(49 Ø Ø 1 Ø Ø) (IMPINGE+) (IMPINGE−)
NIL
*(ADD DELAY DEEP-CEREBELLAR-NUCLEUS
  15$
DELAY4Ø
(INPUT: 3)
*(ADD DELAY N-RUBER 2Ø$
DELAY36
(INPUT: 49)
*(TIE EXCITATORY DELAY4Ø N-RUBER$
N-RUBER
(49 Ø Ø 1 Ø Ø) (IMPINGE+ DELAY4Ø)
  (IMPINGE−)
NIL
*(TIE EXCITATORY DELAY36 FLEXOR-
  ALPHA-MOTORNEURON$
```

```
FLEXOR-ALPHA-MOTORNEURON
(32 Ø Ø 2 Ø Ø) (IMPINGE+ DELAY36 DELAY21
  DELAY5 AFFERENT2-INTERNEURON
  EXT-INTERNEURON1B) (IMPINGE−
  DELAY12 EXT-INTERNEURON1A
  RENSHAW-CELL
INTERNEURON1B)
NIL
*(TIE EXCITATORY DELAY36 GAMMA1-
  MOTORNEURON$
GAMMA1-MOTORNEURON
(34 Ø Ø 1 Ø Ø) (IMPINGE+ DELAY36 DELAY26
  DELAY21) (IMPINGE-INTERNEURON1B
  DELAY 12)
NIL
*(SAVE RUBER$
JOB-SAVED
```

In this example, the user wished to add a new structure to the model, namely nucleus ruber (N-ruber), and to connect it *via* appropriate delays so that it would get excitatory input from the deep-cerebellar-Nucleus (already in the model), and deliver excitatory output to the alpha- and gamma-1 motorneurons on the flexor side. The instruction (ADD N-RUBER) requires specification of the new structure's threshold which was chosen to be 1.

Upon requesting the addition of delays (ADD DELAY with the statement of point of origin and duration), the program returns the call by assigning a number to the new delay (the numbers 40 and 36 were assigned, respectively).

The counterpart to ADD is the instruction DELETE (not shown here).

Once all new structures are added and their thresholds and delay length specified, the user applies (TIE) which calls for three arguments: excitatory (or inhibitory)

name of the structure of origin

name of the structure of endpoint of new link.

The input to the newly added delays is already furnished by the instruction ADD. After the new connection is established, the program lists all excitatory and inhibitory inputs to the recipient structure of the new connection in the form if (IMPINGE+ . . .) and (IMPINGE− . . .), respectively.

The counterpart to TIE is CUT which severs existing connections and requires the same format of arguments as does TIE.

The user then decides to save the new model under the name of RUBER.

Two more changes are then made: a driving force of +2 is added to the Deiters-Nucleus by means of the instruction (TRY DRIVER . . . value). This means that deiters nucleus will now have an output of two in the absence of any excitatory input, and this value of two will be added to any output level computed (with the ceiling value, of course, remaining 4). Furthermore, the sensitivity of the nuclear bag end organs is changed: first, the user examines their old value with KBAG; there are 4 criterion levels set (1 through 4); any length difference between extra- and intrafusal muscle fibers up to 25 generates an output of 1; similarly, the values 2, 3 and 4 are generated by the specified length differences. These criterion values are changed by STORE (KBAG etc.).

```
*(INSTATE RUBER$
NIL

NIL
*(TRY DRIVER DEITERS-NUCLEUS 2$
OK
*(KBAG 1$
25
*(KBAG 2$
5Ø
*(KBAG 3$
1ØØ
*(KBAG 4$
2ØØ
*
 (STORE (KBAG 1) 5Ø$
5Ø
*(STORE (KBAG 2) 1ØØ$
1ØØ
*(STORE (KBAG 3) 2ØØ$
2ØØ
*(STORE (KBAG 4) 3ØØ$
3ØØ
*
```

Neurological lesions can be simulated by the command (KILL—name of structure—) which sets the threshold of this structure to the value of 4 and, thus, effectively eliminates the structure from the flow of activity in the network.

The command BLOCK, decreases the activity level of a set of one or more structures by a specified value: for example, (BLOCK T 1 @(RAS RAS-INTERNEURON)). The neural structures within the parentheses are affected. The converse effect is generated by the command (FACILITATE T +1 @(RAS)) which mimics an excitatory drug effect.

OBSERVATION OF MODEL PERFORMANCE

To enable testing the performance of the model with the types of maneuvers applied by neurophysiologists or neurologists to evaluate the functional state of the motor system, two functions are defined:

> *PULSE,* designed to mimic a brief muscle pull, and consisting in the application of input pulses to the muscle spindle afferents of a specified muscle for a brief, specified period of time. This function is intended to test the motor system for its ability to support a phasic stretch reflex of the kind commonly employed in the form of the "knee jerk."
>
> *STRETCH,* on the other hand, is designed to test for active resistance to muscle stretch, developed in the course of slow extension of the muscle under study, with rate and duration of passive muscle stretch to be specified by the user. The muscle tension developed during passive stretch is computed as the incremental activity required at the Golgi stretch receptors to sustain the specified rate of muscle elongation during the stretch interval.

For the assessment of the model's response to these kinds of stimuli, the important variable to be observed is that of muscle length. In addition, the user may wish to observe activity levels of selected neural structures. He can accomplish this defining a "WINDOW" that lists these structures by name.

Even if only a few structures of the model are being monitored, the output generated as a result of several hundred state transitions in discrete time becomes bulky and cumbersome to interpret; details that should stand out to attract the user's immediate attention, tend to become obscured. Thus, some compression of output is needed to enable quick and reliable "perception" of the performance.

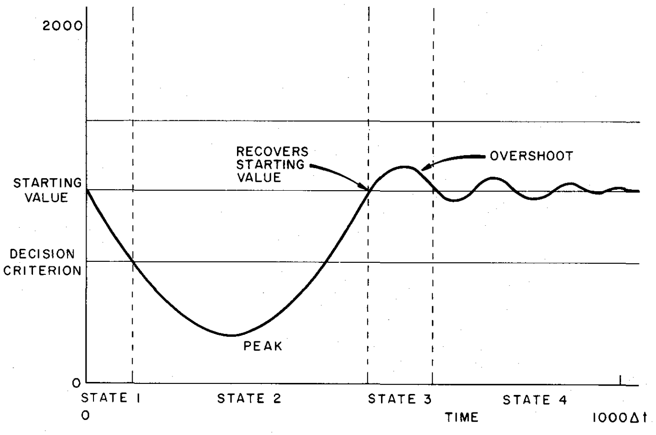


Figure 4

For the purpose of perceiving the characteristic features in muscle behavior following PULSE or STRETCH perturbations, we view the resulting transient departure from resting muscle length as comprising four states (Figure 4):

State 1: commencing with the application of the stimulus and extending until deviation of muscle length exceeds criterion level;

State 2: follows state 1 and extends until resting muscle length is recovered; value and time of occurrence of the extremum, and time till recovery are recorded.

State 3: follows state 2 and encompasses the period during which the primary overshoot occurs; its magnitude is recorded.

State 4: encompasses the remainder of the observation period during which mean muscle length, maximal excursions and frequency of oscillation are recorded.

In the case of what would be considered a normal reflex, the observation terminates in state 4. Abnormal or abortive reflex behavior can cause the perception algorithm to terminate in any of the earlier states, in which case appropriate messages are generated.

For the neural structures named in a WINDOW, the compressed output consists of a mean activity level over the observation period and the number of oscillations that may have occurred. Moreover, as shown below, the algorithm records duration and time of occurrence of the longest sustained periods of high and low levels of activity. Although not illustrated, the program also computes time of occurrence and intensity of the first burst of activity triggered by the test procedure; this reflects the temporal dispersion of transient activity in the neural network.

```
*(SETQ WINDOW @(N-VENTRALIS-
  ANTERIOR GAMMA1-MOTORNEURON))
(N-VENTRALIS-ANTERIOR GAMMA1-
  MOTORNEURON)
*(PULSE FLEXOR 3 50 1ØØØ)

FLEXOR-MUSCLE
STARTING-LENGTH 1ØØØ
EXTREME-VALUE 800, OCCURS-AT 75
STARTING-VALUE-REGAINED-AT 24Ø,
  OVERSHOOT 32
```

```
STEADY-STATE-VALUES: MEAN 999,
  STANDARD-DEVIATION 51 HIGHEST-VALUE
  1088, LOWEST-VALUE 908, #-OSCILLATIONS 28
FINAL-VALUE 1036
DO-YOU-WISH-TO-EXAMINE-WINDOW *YES

N-VENTRALIS-ANTERIOR: MEAN 17, #-OSC 56
  LONGEST-RISE 38, OCCURS-AT 112
  LONGEST-FALL 20, OCCURS-AT 188
GAMMA1-MOTORNEURON: MEAN 11, #-OSC 70
  LONGEST-RISE 20, OCCURS-AT 126
  LONGEST-FALL 26, OCCURS-AT 963
```

While these perception algorithms describe model behavior in absolute and numerical terms, observational records in the natural domain are typically stated in terms of qualitative descriptors, most commonly of a comparative nature; that is: in terms of deviations from a reference state. To enable the modeler to perceive his simulation results in these same terms, we devised the program ANALYZE. This program performs a sequence of tests on two specified states of a model, compares the results and generates a verbal output that describes differences in behavior along certain key dimensions. An application of this program is illustrated below: at first, a lesion of substantia-nigra is produced by applying the KILL-instruction; this causes the behavior of the model to depart from the normal condition in a manner resembling that of Parkinson's disease in that the response to a PULSE stimulus of the flexor muscle is augmented and abnormally prolonged while the STRETCH of the flexor muscle induces an oscillatory response (i.e., tremor).

```
*(KILL SUBSTANTIA-NIGRA)

DONE
*(ANALYZE @NORMAL)

((PULSE-FLEX NEVER-RECOVERS LATE-PEAK
  AUGMENTED-PEAK) (PULSE-EXT ABSENT)
  (STRETCH-FLEX OSCILLATIONS EARLY-
  RECOVERY) (STRETCH-EXT NEVER-
  RECOVERS))
*
```

The tremor component of this pathology is in the next simulation run alleviated by combining the lesion of substantia nigra with a second lesion, intended to mimic one form of neuro surgical management of Parkinson's disease, namely a lesion of the nucleus ventralis anterior of the thalamus:

```
*(KILL SUBSTANTIA-NIGRA)

DONE
*(KILL N-VENTRALIS-ANTERIOR)

DONE
*
 (ANALYZE @NORMAL)

((PULSE-FLEX NEVER-RECOVERS LATE-PEAK
  AUGMENTED-PEAK) (PULSE-EXT ABSENT)
  (STRETCH-FLEX EARLY-RECOVERY)
  (STRETCH-EXT NEVER-RECOVERS))
*
```

REFERENCES

1 D W LOVELAND
Theorem provers combining model elimination and resolution
Machine Intelligence Vol 4 ed B Meltzer and D Michie
American Elsevier 1970

2 J A ROBINSON
A review of automatic theorem proving
Ann Symposium in Applied Mathematics 19:1 1967

3 C S PEIRCE
Collected papers of Charles Sanders Peirce
C Hartshorne P Weiss and A Burks eds 8 vols Cambridge
Mass pp 1931-1958—Especially Vol II pp 272-607

4 W S McCULLOCH
What's in the brain that ink may character
Presented at the International Congress for Logic,
Methodology and Philosophy of Science 1964-Reprinted in
W S McCulloch Embodiments of Mind MIT Press 1962

5 H E POPLE JR
A goal oriented language for the computer
In Representation and Meaning—Experiments with
Information Processing Systems—H Simon and N Siklossy
eds Prentice Hall 1972

6 D ISNER
An inferential processor for interacting with biomedical data using restricted natural language
AFIPS Conference Proceedings Spring Joint Computer
Conference 1972

7 H E POPLE JR
On the mechanization of abductive logic
In preparation

8 J C M ECCLES ITO SZENTAGOTHAI
J SZENTAGOTHAI
The cerebellum as a neuronal machine
Springer Publications 1971

9 R GRANIT
The basis of motor control
Academic Press 1970

10 J McCARTHY P W ABRAHAMS D J EDWARDS
T P HART M I LEVIN
LISP 1.5 programmer manual
The MIT Press 1962

11 L H QUAM
LISP 1.6 manual
Stanford Artificial Intelligence Project

The clinical significance of simulation and modeling in leukemia chemotherapy*

by T. L. LINCOLN

The Rand Corporation
Santa Monica, California

INTRODUCTION

It seems certain that in the next thirty years—by the year 2001—the computer will become a major instrument in support of clinical practice. Part of our task today is to look beyond the frustrations, expense, and apparent waste of our present prototype systems and to distinguish the important themes which will bind medicine and computers together. In computer medicine we are clearly in the exploratory phase of a new technology—a position analogous to the early designers and users of automobiles. With the introduction of the automobile came a great variety of expensive, made-to-order cars which could only be run on poorly maintained muddy roads. To drive required a large measure of patience, enthusiasm, and self-reliance, coupled with a fine sense of the ridiculous. Moreover, the automobile scared the horses, and otherwise challenged the established order of things. And still it came, demanding new conventions to make its use effective: freeways, parking tickets, campers, bedroom communitites, air pollution—the good with the bad.

This is a talk about mathematical modeling, simulation, and leukemia chemotherapy; but I have opened in this way because it is not the details of leukemia chemotherapy which I want to emphasize, but rather the far-reaching changes which information processing is destined to bring to this field. The primary changes will be found in the conventions of clinical decision-making. The use of mathematical modeling and the computer in leukemia chemotherapy is particularly illustrative of a new form of medical practice because (1) the untreated disease is usually rapidly fatal and deserves the investment; there are 14,000 new cases a year, many in children and young people; (2) it is quantifiable; the leukemia cells can be sampled in the blood and the bone marrow; (3) it responds to rational, well-organized therapeutic regimens; effective drug doses and schedules can be set up which depend upon cell kinetics (models of cell behavior) and pharmacokinetics (models of drug distribution); (4) the impact of drug therapy is quantifiable, thus the regimens can be evaluated and the models verified; and (5) good results depend upon a persistent, well-organized approach.

We might put the argument another way: We have strong drugs with which to treat leukemia, but our therapeutic advantage is sufficiently small, and our knowledge sufficiently limited that the penality for sloppy thinking and sloppy patient management is particularly high. This disease is a test of our ability to manage medical information, to understand its implications, and to apply our knowledge wisely. Because quantitative data is the key aspect of decision-making in this disease, skill in the use of quantitative information is essential to success. At the same time, because of the importance of teamwork between the specialties of hematology, clinical oncology, infectious disease, and laboratory medicine, leukemia therapy is an excellent test of the ability of computer technology to coordinate the relevant information. Moreover, success, short of a cure, requires long term sophisticated maintenance therapy, and thus effective long term records. To pursue the automobile analogy, we might consider the skill, the teamwork, and the endurance required for this information processing system to be in some sense parallel to that required to run in the Indianapolis 500.

DISCUSSION

The chemotherapy of acute myelogenous leukemia was chosen for my research as the principal area for modeling. This disease lends itself to a quantitative

* This work is supported under NIH Grant CA-12369.

analysis because there are good measures of success, failure and complications. Treatment is quantitative and offers a spectrum of options in detailed implementation. The purpose of simulation is to make the choice of treatment options more rational. Leukemia can be compared to the growth of crab grass in a lawn. If left by itself, this agressive grass will overtake the lawn and destroy it. In a like manner, leukemic cells grow in the bone marrow and come to replace the cells which are usually present there. As the bone marrow is encroached upon, its normal function—to supply red cells, granulocytes, and platelets to the blood— is compromised. The lack of red cells leads to anemia; the lack of granulocytes leads to an increased susceptibility to bacterial infection; and the lack of platelets—a cell needed for blood clotting—leads to hemorrhaging. If nothing is done, the patient will die quickly from one of those complications. Treatment must stop the process and turn it back. The ideal objective of chemotherapy is to kill the encroaching leukemic cells without injurying the normal cell lines. However, the best that can be done is to establish a differential kill rate based on differences in growth characteristics, so that on balance more leukemic cells die than normal cells. In the crab grass analogy, the use of a growth poison, such as chlordane, achieves a differential kill rate because the crab grass grows more rapidly than the lawn grass. It is generally not possible to irradicate crab grass completely. If treatment is stopped, the crab grass will recur. But it is possible to reduce it to a nearly imperceptible level, and to maintain the lawn with continued intermittent treatment. The best results depend upon adjusting the dose of chemical and the interval between doses to keep the lawn and to kill the weed. Two different strategies are needed: one to get rid of the visible crab grass; the other, to maintain a healthy looking lawn. This healthy looking state is medically equivalent to remission.

In leukemia chemotherapy the first strategy is to achieve remission induction, the second remission maintenance. Modern experience indicates that large doses of drugs given at intervals of about 14 days, which first produce marrow depression and then allow for marrow recovery, give the best induction results and lower doses at longer intervals provide good maintenance. The toxic effects of the drug, by reducing the function of the marrow, lead to the same complications: anemia, infection due to lack of granulocytes, and bleeding due to lack of platelets.

This is the briefest overview of the clinical situation and the short term clinical objectives. Clearly, one would like to develop methodologies which will cure the disease, but in the meantime, the clinician must move between the two sources of marrow disfunction: overtreatment and undertreatment. He has at his disposal few objective measures which might be considered as "control variables."

For example, the concentration of the cellular components in the blood reflect bone marrow function. Daily blood samples can chart not only the impact of the chemotherapeutic drugs, but also the likelihood of complications. Bone marrow biopsy allows the developing blood cell components to be counted and provides an assay for the encroachment on the marrow of leukemic cells and the depletion of the marrow by drug.

Classically, decision procedures based on these data have been organized into clinical rules of thumb—a heuristic syntheisis of experience. For example: "When the platelet count drops to 20,000, there is danger of bleeding, and we give platelet transfusions." These rules of thumb are modified by clinical asides: "The threshold is set a little high on purpose—the patient probably won't get in trouble until his count reaches 10,000."

Computer programs have been written which incorporate such rules of thumb. The best known are those on electrolyte balance. One might say that these programs automate the professor in a dogmatic way and can only be as successful as his dogma. The same can be said for programs which set up trees for sequential decision-making. The output of the program is no greater than the input.

The advantage of a simulation is that one can explore new territory. If the problem has been formulated in a mathematically explicit way so that the interaction among the variables is accounted for, new states of the system can be explored. If the simulation were perfect, then the simulation would be equivalent to the essence of the problem and could serve as a near perfect advisor. However, in a situation as complicated and as incompletely understood as the biology of cancer chemotherapy, we can hardly expect to fulfill this ideal. What then is the purpose of mathematical modeling, of simulation, and of the most expensive luxury of all—on-line interactive graphic simulation?

As a physician working in this field, I can only present my experience with the hope that it offers some generality. To build my models, I have been using BIOMOD, a Rand-developed software package designed specifically for on-line interactive model building and simulation. It provides a data tablet, keyboard, and full graphics capability with a resolution of 1024 by 1024. The system has the capacity of enter models in mathematical formats so that the computer does the machine-language programming and can produce, compile, and run a new program in a matter of minutes. The system can keep track of 20 output curves and display five of these on command. As a simulation progresses, the evolving curves are presented on the screen. The scale of the graphic output can be changed interactively and the displayed variables combined and redefined. The simulations can contain up to 20 modifiable parameters so that the simulation can be stopped, parameter values changed, and then allowed to proceed. In BIOMOD II, the new FORTRAN-based revision of BIOMOD, we now have the capacity to compare the output with data curves which can be entered directly through the tablet.

These capabilities cannot be provided in batch processing. In short, for a physician this is a very fancy system. What are the returns? To present these, I identify four different levels of model building.

1. *Parameter Identification.* The process of trying to build a simulation of a medical problem is in and of itself elucidating. The necessary first step in this exercise, however primitive the quantitative results may be, is the identification of those parameters which ought to be considered in the model. In leukemia chemotherapy we presume that the clinical phenomena can be understood by constructing models which simulate the growth of cells and developmental characteristics of the blood cell precursors in the bone marrow. Ultimately our models must incorporate the following biological processes:

1. The mechanisms by which blood-forming tissues sustain themselves.
2. The control mechanisms that regulate the growth and differentiation of particular blood-cell types.
3. The relationships between the blood as a circulating pool of cells and the other tissue spaces where blood cells are sequestered.
4. The processes of attrition that lead to the natural loss of cells from the blood.
5. The ways in which cytotoxic drugs act on blood-forming tissues (e.g., the sensitivity of cells in different phases of the cell cycle to particular drugs).
6. The rates of transport and the detoxification of particular drugs.
7. Growth characteristics which distinguish leukemic cells from normal cells.

This biological analysis gives focus to the data items which can be collected in a clinical setting. Some items are collected to evaluate parameter values. For example, tritium thymidine studies can evaluate cell kinetic characteristics. Some are collected to follow the evaluation of certain control parameters, for example, the platelet count. Given even a simple model of platelet destruction, it is possible to observe when the count is falling as a function of the platelet pool only, i.e., when no platelets are being made. Now the clinical decision rule can become an anticipatory one: "If the halving time of platelets is under 36 hours, then prepare to give a platelet transfusion before the count falls to 10,000." The result, at the very least, is a data flow chart with explicit organization and meaning, very much in the spirit of Larry Weed's automated record. Manual and now automated flow charts have come into use at the M. D. Anderson Hospital to monitor the changes in important variables. These allow a chief of service to review thirty patients in less than two hours rather than by an exhausting set of card rounds. Such flow charts can condense the information both numerically and graphically. The prototype of the graphic representation is the temperature chart, long clearly useful. The updatable graphic record of lab value changes over time, coupled with important therapeutic decisions, provides a scanable situation report which should come to replace the chaos of the classical record.

2. *Functional Organization.* A step beyond the assertion that a certain group of parameters are important and somehow related is an explicit graph of this relatedness. In the case of the blood and the marrow, this is illustrated by Diagram I. Such a block diagram does not describe the exact relation among the variables. Rather it sets the stage for their discovery through the organized analysis of experimental and clinical data. Mathematical models of bone marrow cell kinetics and leukemic cell kinetics have been formulated to conform to the available clinical and experimental data. The models that we have been building depend for their

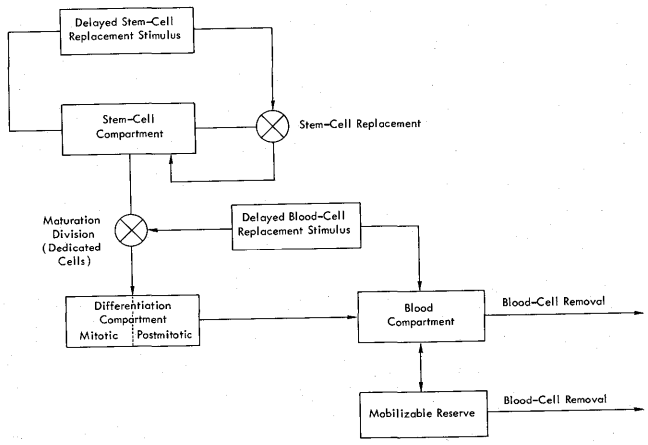

Delayed Stem-Cell Replacement Stimulus
Stem-Cell Compartment
Stem-Cell Replacement
Maturation Division (Dedicated Cells)
Delayed Blood-Cell Replacement Stimulus
Differentiation Compartment
Mitotic
Postmitotic
Blood Compartment
Blood-Cell Removal
Mobilizable Reserve
Blood-Cell Removal

Diagram I.

clinical input on the treatment protocols and the resultant data from the adult leukemia service of the Department of Developmental Therapeutics, M. D. Anderson Hospital, University of Texas, Houston, Texas. Here the greatest pay-off is ins'ght, however incomplete. For example, we observe clinically that the platelet count rises two to three days before the granulocyte count, when a heavily treated marrow is undergoing recovery. If the granulocytes are dangerously low, the platelet count is a good predictor of how long the count rises two to three days before the granulocyte count, when a heavily treated marrow is undergoing recovery. If the granulocytes are dangerously low, the platelet count is a good predictor of how long the white cells will be depressed. The relationships which outline the flux of cell types through the bone marrow provide us with this qualitative observation. In biological simulations it is important that the system can handle time delays and other processes which exhibit historical dependence. For example, the attrition rate of red cells is proportional to the age of the cells, with an expected life-span of 120 days. Thus, a marked variation in cell production influences the number of cells present at a much later date. BIOMOD incorporates efficient means of handling delays and some kinds of historical dependence.

3. *Teaching Model.* These simulations exhibit properties which are thermatically correct. In our case, it is the typical response of a typical leukemia patient to a typical course of drug therapy, either for remission induction or remission maintenance. At this stage interactive intervention in the evaluation of a simulation becomes useful and of insight. There is a natural inclination for physicians to extrapolate a little bit ahead of the available results. Thus the ability to watch a simulation unfold allows learning, model correction, and experimentation. At present, we can only project the clinical impact of such a system. We observe that it allows an exploration of therapeutic alternatives in a very real and believable Socratic Dialogue. It is also well to observe that it presents a danger, which might be called the "Las Vegas effect": There is a fascination with on-line simulations which can eat up computing time and money, the implication being that some form of success is just around the corner.

Our present modeling efforts are aimed primarily at the teaching model level of sophistication. On the biological level, teaching models test the consistency of our hypotheses. For example, present models point to a major discrepancy between our theoretical understanding and our practical results. The models suggested by the work on the L-1210 leukemia in mice and first outlined by Skipper have led to considerable clinical success. In mice the differential kill rate of leukemic cells over normal cells depend on constant differences in the cycle time, the growth fraction and the length of the DNA synthesis phase between the L-1210 cells and the mouse marrow. The experimental tumor grows faster, all cells are actively dividing, and are thus more susceptible to cycle sensitive drugs like cytosine arabinos'de. Very successful schedules have been set up assuming similar cell kinetics for human leukemics. However, in the latter case, Clarkson has demonstrated that in relapse, with a bone marrow full of leukemic cells, the leukemic cycle time is longer than that of normal cells, and the growth fraction smaller. This points to a new modeling requirement, a new iteration: the analysis of transient growth states leading to a new hypothesis for the same output phenomenon.

We see that the clinical pay-off from a teaching model is *clinical courage*, the willingness to carry through a plan because, in spite of limitations in the underlying rationale, the phenomena have become more or less predictable. Modification must be made for the individual patient using classical clinical judgment.

4. *Patient-Specific Model.* Whereas the teaching model provides a skeleton for action, the patient-specific model demands that parameters be modified to take into account individual differences. If this can be done successfully, i.e., with clinical verifiability, then the modeling process becomes a component in the therapy itself, a medical tool like any other. In cancer chemotherapy we have not taken this step, and except for very simple model components, this form of modeling remains largely a vision of the future. However, we can see it as a vision of the immediate future, because at least one strategy for implementation already exists: at a given point in time, the measure of a biological response of the patient can be derived from the data accumulated over the past course of therapy. Thus, an initial protocol course of drug has the dual objective of therapy and a calibration of future therapy. We envision combining protocol data curves with our simulations, using subroutines to automate the evaluation of parameter values.

This approach should open up new potentials in protocol studies of cancer chemotherapeutic drugs. Present protocols treat patients by a formula which supposes that each is the average case of the teaching model. Drug courses are set up so that cons'stency is achieved by using standard doses. This imposes certain obvious rigidities which limit success. By using a model approach, it will be possible to keep the relationship between drugs and patient parameters constant. Thus we can seek to test the input of drugs on the biology of the patient and his disease, rather than treating him as a block box, susceptible only to statistical analysis.

CONCLUSION

The impact of model building and simulation on clinical medicine is just beginning to be assayable. Many of the themes which will ultimately be important can be observed in prototype. Long-term effects will be felt (1) on the operational organization of medical data, in the medical record; (2) on the formulation of the biology of disease; (3) on the methodology of medical teaching; and (4) on the precision and focus of clinical research.

REFERENCES

1 H L BELEICH
Computer-assisted evaluation of electrolyte and acid-base disorders
Journal of Clinical Investigation Vol 49, p 10 1970

2 G F GRONER R L CLARK R A BERMAN E C DELAND
BIOMOD: An interactive computer graphics system for modeling
The Rand Corporation R-617-NIH July 1971

3 L L WEED
Medical records, medical education and patient care
The Press of Case Western Reserve University Cleveland Ohio 1969

4 E GEHAN E J FREIREICH
Department of Developmental Therapeutics M D Anderson Hospital Houston Texas Personal communication

Graphics software for remote terminals and their use in radiation treatment planning*

by KARL H. RYDEN and CAROL M. NEWTON

University of California at Los Angeles
Los Angeles, California

INTRODUCTION

Interactive graphics' ability to provide a meaningful interface to investigators in a variety of applied fields has been recognized for many years. Nowhere is this promise greater than in medicine and the life sciences. Levinthal's[1] interactive program for constructing and displaying complex molecular structures is well-known. Neurath[2] and Frey[3] have analyzed chromosome spreads by delegating to human perception the light-pen dissection of individual chromosomes from overlapping clusters while leaving subsequent measurements and final classification to the computer. Cox and Clark's Programmed Console[4] has introduced many radiotherapists to the use of computers in treatment planning. Other applications at this facility[5] and elsewhere[6] illustrate the great value of interactive graphics support in deriving insight from large clinical data bases, exploring complex biological models in the hope of improving treatment strategies, and developing cost-effective algorithms or special hardware for patient monitoring.

That the use of computer graphics in biomedical and other fields has not developed more rapidly has been in large part attributable to two factors, the high costs associated with hardware and development of applications programs, and the general unavailability of graphics terminals that can operate remotely using common carrier communications facilities. The latter capability is not needed when modest computational requirements enable support by a dedicated small computer, as in the case of the widely used Programmed Console. But, as illustrated by other of the projects mentioned above, many biomedical applications require access to a level of computer support which is most economically provided on a shared basis. The recent commercial availability for less than $20,000 of a terminal (IMLAC) which has many characteristics of the higher quality graphics systems and which uses common carrier telephone facilities to communicate with its host processor, has motivated a substantial investment in systems support at UCLA's Health Sciences Computing Facility (HSCF).

The setting for this work will be described, previous efforts in graphics and the development of an operating systems environment appropriate for supporting a multi-terminal remote graphics system. Subsequent discussion of a set of interactive graphics programs that are being developed for radiation treatment planning will illustrate some of the problems that must be faced when a graphics terminal is to be supported by low-bandwidth communications. After describing the related hardware, current and projected software developments will be discussed with reference to the foregoing applications.

PREVIOUS DEVELOPMENT OF GRAPHICS AND OPERATING SYSTEMS AT HSCF

HSCF is maintained by the National Institutes of Health to explore and develop computer-implemented analytical support to research in biology and medicine. The BMD statistical package programs, TORTOS operating system, and other developments at this facility have originated in response to explicit needs of biomedical researchers.

TORTOS (**T**erminal **O**riented **R**eal **T**ime **O**perating **S**ystem)[7] was developed in response to the variety of needs that prompt biomedical researchers to share access to a large computer. It has been designed to take

* This work is being undertaken at the Health Sciences Computing Facility, UCLA, sponsored by NIH Special Research Resources Grant RR-3.

advantage of the large high-speed core (2000K bytes) and computational speed of the 360/91 processor, making these resources available to the facility's users with as few constraints as possible on the types of programs accommodated. The terminal user has at his disposal the full facilities of a standard OS/MVT system as well as the specialized interactive functions developed here to aid conversational access to the system (monitor programs, a convenient file service, input/output interfaces, text editor, a FORTRAN compiler, and a library of statistical and biomedical applications programs). The system concurrently supports local and remote batch operations. Its versatility is exemplified by the wide range of uses to which it is being put. These include multivariate statistics, computer-aided instruction, management information systems in the Biomedical Library, biological modeling, and several clinical applications. Although not presently in use, the system has supported a high speed, real time link to an SDS 9300 located in UCLA's Brain Research Institute.[5] Since the remote graphics terminals are time-shared as an extension of the conversational terminal system, they have access to all facilities developed for the latter.

For the past five years, we have developed and use-tested systems to make the IBM 2250 graphics terminal accessible to biomedical programmers. Interactive graphics interfaces are intended to be meaningful to the scientists who use them. A growing number of biologists and physicians are capable FORTRAN programmers. Many of this facility's medically or biologically oriented graphics programs have been developed by such people, using our FORTRAN callable subroutines, GRAF (**G**raphic **A**dditions to **F**ORTRAN)[8,9] or a similar interface, PL/OT,[10] developed for use with PL/1. These include[5] programs for data analysis and statistics, teaching, image processing, modeling of cellular and physiological systems, interactive retrieval, simulation of disease propagation in adjacent urban environments, design of coronary-care and obstetrical monitoring systems, and radiation treatment planning.

The GRAF programmer has at his disposal routines for generating the primitive elements of an image (blanked and unblanked points, lines, and alphanumeric characters) which are combined into a structured display file whose basic element is called a display variable (DV). Additional routines are provided for combining the DV's into display frames for presentation on the terminal's screen. There are also facilities for servicing the input devices associated with the graphics terminal (alphanumeric and function keyboards and light-pen) and for character-string manipulation.

REMOTE GRAPHICS SUPPORT TO RADIATION TREATMENT PLANNING

Distributed-processing capabilities being developed for the GRAF subroutines designed for terminals operating over low-bandwidth communications systems are discussed in later sections. One of the biomedical projects whose needs influence this work, a set of programs for radiation treatment planning, is described here. Preliminary program development on the IBM 2250 has been reported.[11]

Previous work in radiation treatment planning

Radiation treatment planning is among the earliest established areas of computer support to medical practice. Stovall's[12] recent bibliography provides access to most of the formal publications in this field and is recommended as a basic reference for this section.

The radiotherapist's primary concerns in treatment planning may be briefly summarized as follows: He has at his disposal radioactive sources that can be placed within the body and beams of radiation that can be directed to it from various angles outside. He seeks a combination of either, or sometimes both, so that the resulting spatial distribution of dose delivered throughout the tumor volume is large compared to that delivered to normal tissues he wishes to spare. He also desires a relatively homogeneous distribution throughout the tumor volume itself. Fear of tumor regrowth in undertreated areas, "cold spots," dominates the latter concern, but excessive tissue breakdown in hot spots also is to be avoided. Either formulas inherited from research on the interaction of radiation and high-energy particles with matter, or tabulated empirical measurements of fields surrounding sources or beams in water or other materials of known composition, enable him to compute the desired dose distributions. Palpation, scanning or other techniques are used to estimate the location and extent of the tumor. Special radiographic techniques such as tomography help to locate the boundaries of internal organs to be spared or whose tissue densities are to be taken into account when the dose distributions are calculated.

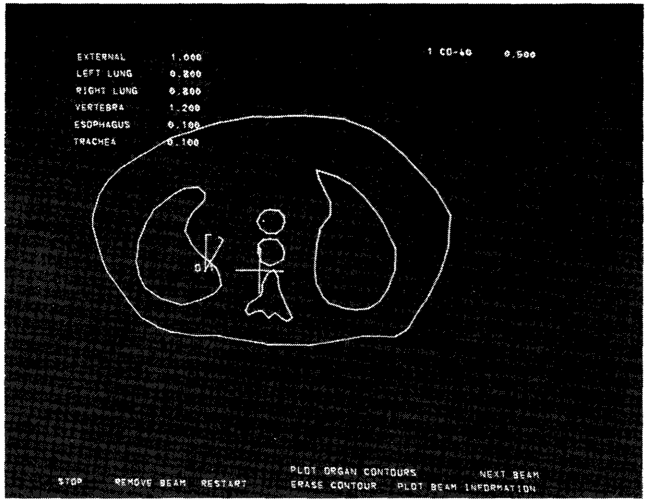
When done by hand, even the simplest calculations, such as the dose distribution for two or three superimposed beams in a homogeneous unit-density medium, are extremely tedious and tend to be undertaken infrequently and reluctantly where computer support is not available. Thus in many places radiation therapy adheres to longstanding conventional prescriptions that have proven relatively effective for the "average patient" given the constraint that the risk of notable side effects in all is to be vanishingly small. Computer-assisted treatment planning seeks to individualize therapy, to help the therapist take every possible advantage of a particular patient's tumor location and body geometry. It also encourages him to explore new general treatment strategies which might, even in places that do not use computers, become part of the conventional armamentarium of treatment plans.

Two types of computer support for treatment planning are regularly used in a small but noteworthy number of treatment centers.[12] Some excellent *batch programs* for external-beam therapy have been tested by many years of use. Representative of these are Theodor Sterling's programs (University of Cincinnati and Washington University, St. Louis), Jack Cunningham's (University of Toronto), and those developed under the direction of John Laughlin (Memorial Hospital, New York). The latter are widely available via remote typewriter terminals. The earliest program extensively used for implant therapy (placement of sources within the body) was developed by Robert Shalek and Marilyn Stovall (M. D. Anderson Cancer Research Hospital, Houston). Both the three-dimensional complexities of implant therapy and the corrections for tissue inhomogeneities found in the batch programs for external beams require major processors.

The Programmed Console developed by Jerome Cox, Wesley Clark, and their associates (Washington University, St. Louis) provides clinicians an economical *interactive graphics* system for treatment planning. It permits simple beam superimposition in a homogeneous body whose outer contour is specified to the computer by a rho-theta digitizing scribe. Program modifications are relatively difficult to introduce because of full utilization of the small processor dedicated to the system, and a number of the capabilities available in batch programs understandably are not present. However, the Programmed Console has demonstrated by wide acceptance the radiotherapist's strong preference for a hands-on graphics system which provides him rapid graphical dose distribution feedbacks for a sequence of conveniently repositioned superimposed beams. Other small-processor graphics systems have been developed in the United Kingdom and elsewhere.

*Remote graphics radiation treatment planning***

Clearly, one wishes to combine as economically as possible the advantages of interactive graphics and major processor support, developing programs having the greatest possible degree of exportability. IBM 2250 programs utilizing GRAF have run on a variety of IBM 360 computers operating under OS, and the new version of GRAF designed for the IMLAC terminal has been organized and will be documented in a manner that should facilitate its rewriting for other processors. Immediate exportability is enabled by the IMLAC terminal's low price and its ability to operate over conventional telephone lines.



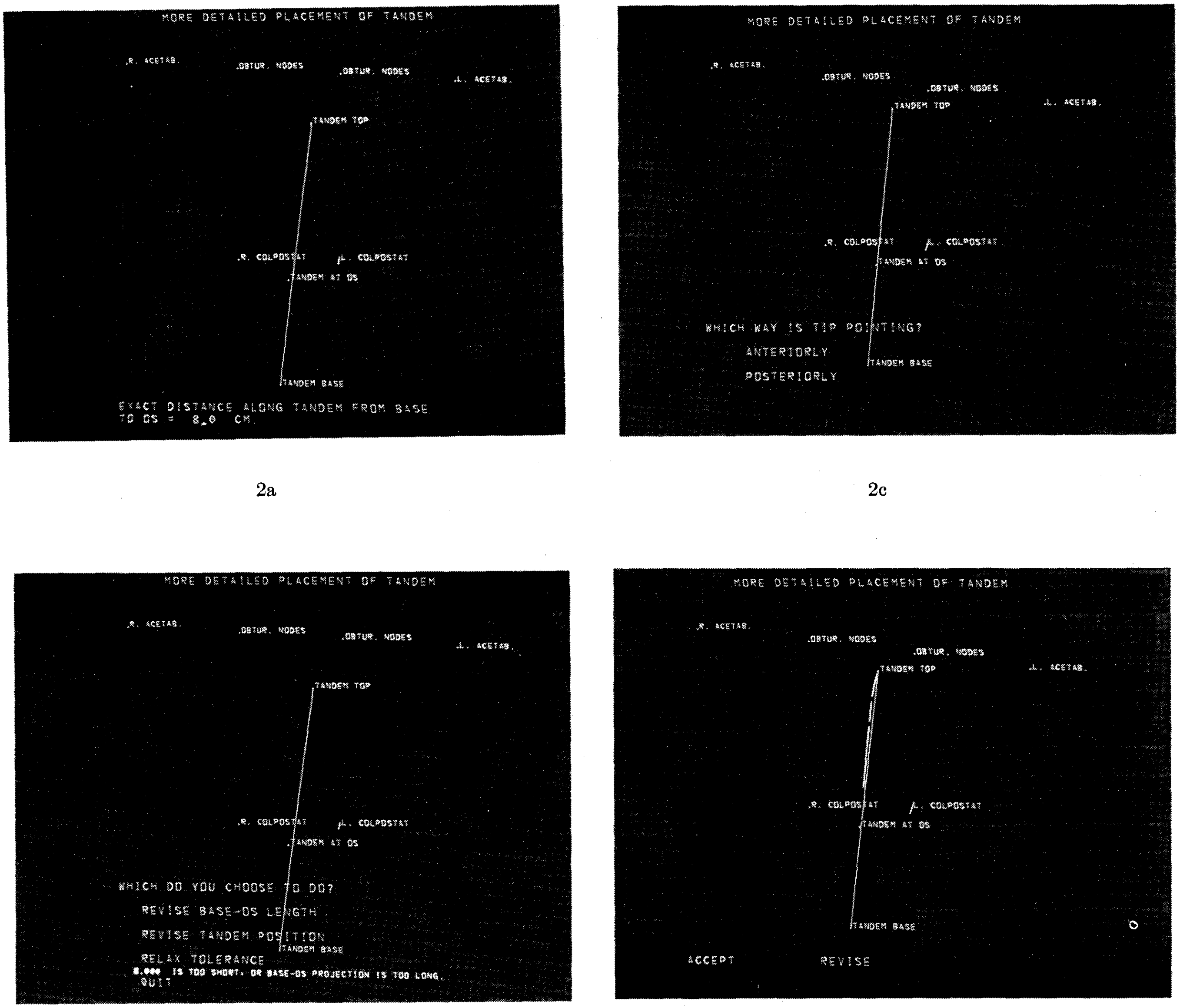
```
EXTERNAL     1.000                1 CO-60    0.500
LEFT LUNG    0.200
RIGHT LUNG   0.200
VERTEBRA     1.200
ESOPHAGUS    0.100
TRACHEA      0.100
```

Figure 1—Treatment prescription display for external beam therapy: Light-pen tracing of organ contours has been guided by a transparency taped to the graphics scope. Information has been entered for the first beam, a Co-60 beam whose center and range of oscillation is indicated by the angular structure to the left of center

The first two members of our package of graphics programs to support radiation treatment planning have been selected as follows: First, one wishes to take fullest possible advantage of excellent batch programs that have been tested by years of use. The initial program in this category is a graphics adaptation of Memorial Hospital's external-beam program. RADCOMP III, the most recent version of M. D. Anderson's program, and Sterling's program will be next. Although there is some overlap among these, it is desirable to offer the different programs that have won various radiotherapists' confidence. Second, we want to explore the special advantages of our system. For this we have chosen a three-dimensional implant application, the widely used Fletcher-Suit intracavitary method for treating cancer of the uterus.

Several capabilities are common to all programs: input of a brief information record for each patient, routines for providing a proper scale and assessing its

** We wish to acknowledge the very valuable collaboration of Richard Nelson, City of Hope, Duarte, California.



MORE DETAILED PLACEMENT OF TANDEM

EXACT DISTANCE ALONG TANDEM FROM BASE
TO OS = 8.0 CM.

2a

MORE DETAILED PLACEMENT OF TANDEM

WHICH WAY IS TIP POINTING?
ANTERIORLY
POSTERIORLY

2c

MORE DETAILED PLACEMENT OF TANDEM

WHICH DO YOU CHOOSE TO DO?
REVISE BASE-OS LENGTH
REVISE TANDEM POSITION
RELAX TOLERANCE
8.000 IS TOO SHORT, OR BASE-OS PROJECTION IS TOO LONG.
QUIT

2b

MORE DETAILED PLACEMENT OF TANDEM

2d

Figure 2—Portion of the interactive specification of afterloader location in Fletcher-Suit treatment of uterine cancer: (a) Important points on the afterloaders having been located on an anterior-posterior (A-P) view of the pelvis, one of the known distances in 3-dimensional space is being requested. (b) An inconsistency being noted between that distance and previous specifications, the user is given the choice of relaxing tolerances or correcting one of the inputs. (c) The computer finally accepts the revisions and asks for the one item of information now required to fully determine tandem location. (d) The sequence of linear sources "loaded" into the tandem in a previous frame is now displayed as it should appear on the X-ray. If this visual check is satisfactory, the user may select from several levels of redundancy in specifying information from a lateral view

uniformity throughout the CRT display whenever graphical information is to be specified by light-pen, and contour plotting routines for displaying dose distributions.

1. *The external-beam program.* When users wish to take advantage of the Memorial Hospital program's capability to correct for tissue inhomogeneities, they must specify contours and densities for the various organs. They may do so by taping a transparent drawing over the CRT face, using the light-pen for a point-

to-point specification of each contour. (A continuous light-pen tracking capability is available on the IMLAC and on some other IBM 2250 models.) It is easy to revise or restart contours. The expense of additional input scribing equipment is avoided. This is much simpler to execute and easier to verify visually than providing the numerical contour specifications required for the batch program. Further burdens of coding input information are relieved as the user next responds to a conversational program that requests information concerning the beams to be used and how they are to be directed (i.e., from a given angle, rotating about a given point, or oscillating within specified angular limits). Brief notations for each beam are added to the body-contour diagram (see Figure 1). Dose-distribution in the plane containing the beams then is displayed. This may be filed for later comparison with other treatment plans.

2. *The intracavitary program for uterine cancer.* In afterloading approaches, hollow instruments (the afterloaders) are placed securely at various locations in the tumor area. Subsequent introduction or removal of radioactive sources within the lumens of the afterloaders may therefore be accomplished without further discomfort to the patient or disturbance of the tumor-source geometry. The Fletcher-Suit afterloading system for uterine cancer comprises a long curved tube which is introduced into the uterine cavity (the tandem) flanked laterally by two source holders (the colpostats) externally adjacent to the uterus. Each of the latter can hold a single radioactive source, and a sequence of sources can be spaced along the central axis of the tandem. Frontal (A-P) and lateral radiographs are used to locate the afterloaders.

A conversational program facilitates light-pen specifications of afterloader locations, taking full advantage of their known geometries. The user may elect tradeoffs between rapidity of input and the security of checks enabled by obtaining redundant geometrical information (see Figure 2). He also may control economy vs. quality of computation by such means as specifying the length of segments to be used to approximate linear radiation sources of uniform density. For instance, an economical initial exploration which approximates each linear source by three segments can be followed by a computation of higher quality which assumes a segmentation of nine or ten. Dose-distribution displays may be requested for any A-P, lateral, or transverse plane (see Figure 3). Inventories are maintained for the afterloaders and sources.

Arrangements are being made for live demonstration of these programs on the IMLAC terminal, in addition to motion-picture illustrations of their use.

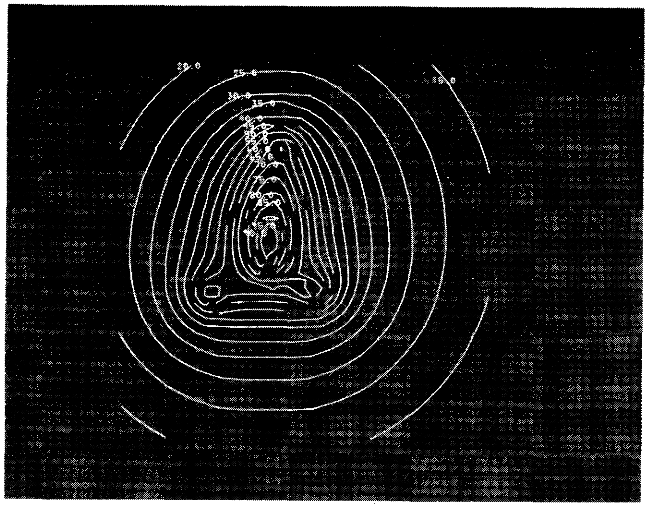


Figure 3—Isodose plot for uterine treatment: Dose distributions may be requested in any A-P, lateral, or transverse plane. This display may be transmitted line-by-line as it is being computed. An occasional segment may be missed by the economical algorithms used here

Possible implications of graphics in the acceptance of treatment-optimization programs

Linear programming[13] and other approaches[14,15] have been suggested for optimizing spatial dose-distributions, and one[14] is being used regularly in a small number of radiotherapy centers in Great Britain. Graphical demonstrations tend to be more convincing than mathematical proofs to the average clinician, and they help him to bring his experience to bear on criticizing the outcomes of using various criteria for optimality. One therefore anticipates that the present conjunction of an interactive graphics terminal with a processor capable of performing optimization calculations will encourage serious investigation, improvement, and perhaps eventual acceptance of more optimization procedures in radiotherapy.

Implementation over low-bandwidth communications systems

Transmission over conventional telephone lines poses serious response-time problems for the more complex graphical displays of organ contours and dose-distribution. Subsequent recall for display purposes rather than initial point-to-point specification is of main concern in the organ-contour display. However, methods currently

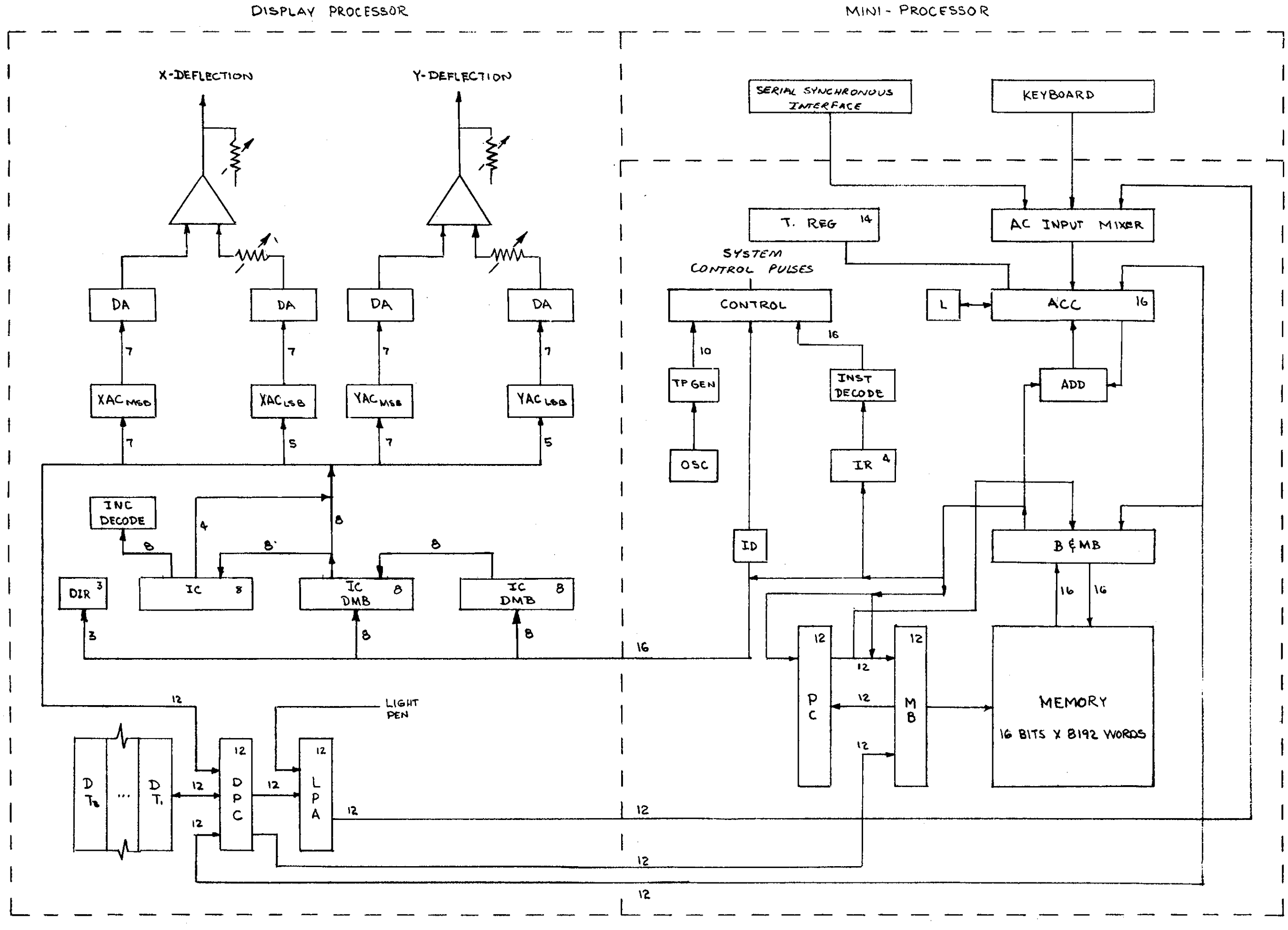


Figure 4—IMLAC PDS-1 system block diagram (redrawn with permission of the IMLAC Corporation) indicating the dual processor nature of the systems design

being explored to expedite transmission of a retrieved display may simplify the initial input as well. Advantages of the interactive approaches that are possible for anatomical specifications or the input of graphical field-distribution data for sources and beams cannot be realized for the computed dose-distributions, since the latter are initiated at the host processor. Research on these problems in graphical representation is being pursued initially on the IBM 2250 while the remote terminal version of GRAF is being developed and tested on the IMLAC. However, both efforts assume a distribution of processing between the terminal and host computer which has not been feasible on the IBM 2250. This work is described in a later section, and additional presentations will be made at the time of the conference.

REMOTE GRAPHICS HARDWARE

The remote graphics system consists of three major hardware components: the host processor, a 360/91 computer with 2 million bytes of high-speed core and 672 million bytes of secondary direct-access storage; the graphics terminal, a mini-computer (PDS-1) manufactured by the IMLAC Corporation; and the data link between the two processors. While the typical graphics program does not normally require the resources of a computer like the 360/91, it does permit the computer to simultaneously service far more graphics terminals than could be handled by a less powerful machine. In addition, the occasional heavy demands for arithmetic processing can be absorbed without severely impacting other concurrent activities, for as at most computer centers, graphics is only one aspect of the computer's total load at HSCF.

The development of a remote graphics terminal is not a new idea, nor is using a small computer to overcome the restrictions of low-bandwidth data links, as has been done elsewhere.[16–18] There are also available today at least five different storage-tube terminals which have a serial-communications interface that permits them to operate from a remote location without a mini-computer. Storage tubes do not need local refresh memories and their display capacity is limited only by the resolution of the screen. By contrast, the display capacity of refreshable tubes is limited by the size of the local refresh memory and the drawing speed (refresh cycle) of the deflection system. However, the storage-tube terminal's limited capacity to selectively erase one component of a display from the screen requires compromises between response time and the ideal presentation of information to be displayed. In order to modify one component, the screen must be erased and the entire modified display retransmitted to the terminal. In one storage-tube system it is possible to erase information from the screen by retransmitting the orders which created the display, but this also erases other components of the display frame which may have intersected the portion which was erased. Lack of a selective erase can be overcome to some extent by the use of a local computer which can regenerate the image; this, however, nullifies much of the cost advantage which the storage-tube terminal otherwise enjoys. The GRAPHIC-II system[16] developed by Bell Telephone Laboratories is similar in many ways to ours. There are, however, some major differences in approaches to optimizing the use of the data link and to distributing tasks between the remote and host processors. The PDS-1 is one fourth the cost of the PDP-9 system used in GRAPHIC-II. In part, this reduction in cost is a result of technological advancement; however, much of it is due to the fact that the PDS-1 was designed specifically as a low-cost display computer.

The IMLAC PDS-1 (see Figure 4) computer is a dual processor machine with each processor sharing the same memory. The main processor executes an instruction set which is typical of most small computers (ADD, SUBTRACT, BRANCHING, LOGICAL and INPUT/OUTPUT operations) while the display processor, which is controlled by the main processor, executes a set of display instructions that generate the image on the CRT (see Figure 5). The display image is drawn using incremental vectors; each incremental order (8 bits) can move the CRT beam in one of 32 directions for a distance of 0-9 raster units (a raster unit is 0.011 inches). Line segments, characters and other graphics figures are generated by display subroutines composed of incremental orders. An eight-level hardware push-down is provided for nesting display subroutines.

The principal disadvantage of the incremental drawing technique used by the IMLAC is the slightly jagged appearance of lines or curves that results from the 32-direction restriction on elements composing them. This can be a serious problem if high-quality images are required. For instance, depth cues of the perspective transformation might be blunted when three-dimensional images are being generated. However, in the majority of medical applications, this is not a serious problem and the economies that result from the use of incremental drawing seem to outweigh the disadvantages.

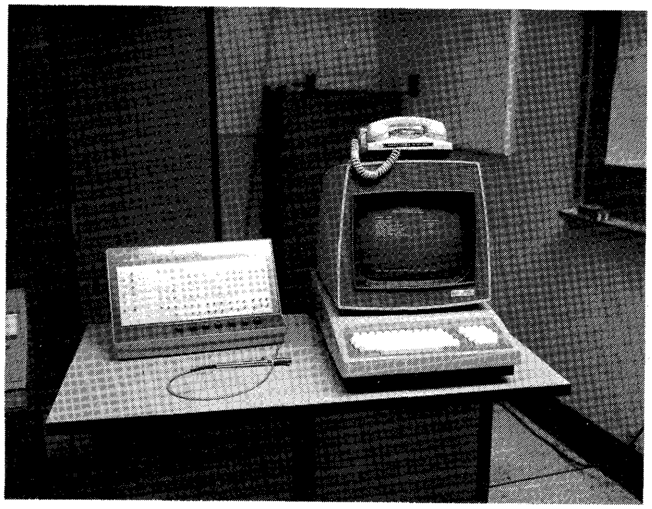


Figure 5—IMLAC graphics display terminal: Light pen and optional programmers console are shown to the left of the keyboard and display

The choice of a data link between the graphics terminal and the host processor was dictated by universal availability, low cost of modems and interfaces, and speed, in that order of importance. We chose to use a serial-synchronous communications format employing switch network telephone facilities at 2000 bps (WE-201A type modems). However, while meeting our requirements of availability and low cost, the transmission speed represents what can only be considered a minimal level. A major component of the programming support is devoted to overcoming deficiencies of the data link.

REMOTE GRAPHICS SOFTWARE

Programming systems support for the remote graphics terminals consists of a control program for the terminal computer, the host processor control program, and the applications programming "language." The latter two components are 360/91 programs while the first executes in the PDS-1 computer. Application programs are written in FORTRAN using a collection of subroutines called GRAF[9,10] which were originally designed for use with the IBM 2250 and whose external format has been preserved in the remote graphics implementation. Thus, use can be made of the large number of programs written using GRAF at UCLA and elsewhere over the past four years. Although an emulation of the 2250 would have provided access to more existing programs, it would have compromised effective use of the data communications facilities.

Strategies for overcoming the limitations of low-bandwidth communications systems

Transmission over conventional telephone lines poses serious response problems for the more complex displays (see Figures 1 and 3). In Figure 1, the organ contour display, the main concern is with the subsequent recall of the image rather than its initial point-to-point specification whereas in displays like Figure 2, response is poor because the terminal fails to react quickly to simple interactions. The logical and arithmetic capabilities of the remote processor offer several means of improving response. Data compression can increase the information density by removing extraneous bits from the graphic data structure or bits whose meaning is implicit from the context in which they are used. Another method is to organize the graphics data so that inactive static segments of the display are saved and recalled into the active display by the transmission of a single command. Labels, grid lines, indexes, and explanatory text are examples of data which are fixed throughout the course of the typical graphics program and need only be transmitted once when sufficient remote processor storage is available. Response time is determined by the variable display data.

Approaches that decrease the number of bits transmitted to specify graphical or other data will be discussed in the next section. The portions of GRAF initially developed for remote IMLAC terminals have emphasized storage-management and look-ahead capabilities that facilitate terminal response by avoiding retransmission of graphical information that may be displayed repeatedly and by anticipating transmission of that information which is most likely to be used next.

Unless the remote computer is augmented by a peripheral storage device, the static display data which can be retained generally will not exceed two or three display frames. The added cost of direct-access storage caused us to prefer extending the storage capacity of the standard terminal by formulating a procedure which retains those inactive segments of the terminal's display buffer which are most likely to be reused when it becomes necessary to overlay some segment in order to accommodate a new segment. This procedure is primarily a storage management technique and has been augmented by look-ahead facilities which are designed to anticipate the requirements for new display segments and transfer them during intervals when the data link would otherwise be idle.

New GRAF storage-management facilities

Implementation of the storage management and look-ahead facilities is largely unknown to the GRAF programmer although he has some control over the transmission of graphics data to the remote processor.

As in the IBM 2250 version of GRAF, the basic element of the graphics data structure is the display variable (DV), which is composed of the primitive elements that generate the display. The DV is formulated in the host processor by graphics subroutine calls, for subsequent transfer to the display buffer, that portion of the terminal's memory which services plotting.

The display buffer of the remote processor is divided into primary and secondary display lists. The secondary display list entries correspond to a subset of the DV's in the host processor. The primary list consists of pointers (display subroutine jump instructions) to each entry in the secondary list which is currently being displayed on the CRT screen. DV's are added or removed from the screen by adding or removing appropriate pointers in the primary list. A display frame is composed of one or more DV's which may contain their own screen coordinates, or the position which the DV occupies may be determined by positioning orders in a DV which "calls" it. The DV can be considered a display subroutine, and in the instance of a relative DV (one which does not contain absolute positioning orders), only one copy of the DV is stored within the remote graphics processor.

The best response characteristics will be obtained when each frame is segmented into DV's such that the static information is separated from the variable data, and when display components which are used repeatedly are each declared to be unique DV's rather than combined into a few DV's. Thus, the user can improve response characteristics not only by partitioning each display frame into fixed-format and variable DV's but also by further subdividing the fixed-format portions to take advantage of relative DV's that are to be used repeatedly. For example, in a single session with the computer, the radiotherapist probably will explore up to three or four treatment approaches for each of several patients. There will be delays when DV's guiding conversational input are initially called, but use of the program thereafter would, for the most part, flow as freely as on the IBM 2250. The saving is especially notable for the grids supplied to effect input of graphical data. One coarse grid covers the entire screen, but for more precise description of curves a finer grid patch is moved to desired locations on the large grid. Treating the patch as a relative DV, only the transmission of two coordinates is required to reposition it. Because of frequent use, DV's representing both types of grid would have a high probability of remaining on the secondary list of the terminal's display buffer.

All of the decision making and tables for managing the remote processor's display buffer are provided by the host processor. The graphics processor's storage management functions are limited to unpacking the transmission records, relocating addresses, and moving entries within the primary and secondary lists. When a new DV is to be transmitted to the remote terminal, its storage requirements are determined by the host processor and it is assigned a relative location within the display buffer. If sufficient free space is available either in one contiguous block or in multiple free blocks, the DV is placed in this region. When a single region is not available, commands to compact the display buffer are transferred ahead of the graphics data. Normally, all free space will be in one continguous block, since the host processor's storage management routines use idle time on the data link to transfer the commands necessary to compact the secondary display list. Upon receiving the DV, the remote processor unpacks the data and relocates all relative addressing information into absolute display buffer addresses. When members of secondary display lists are moved from one location to another to compact the list, all absolute addressing information in both lists is again relocated.

When insufficient free space exists for a new DV that is to be transferred to the terminal, the required space must be created by some combination of using free space and removing secondary display list entries. The criteria used for choosing which blocks(s) in the secondary list should be deleted attempt to reduce the chance of deleting DV's which could be used in a subsequent display while also minimizing response time, since further action by the terminal user is suspended until a successful transfer enables the DV to become part of the active display. Three decision levels are used: (1) The system first tries to find a contiguous set of free or inactive blocks within the secondary display list. The inactive DV's which are chosen are those with the lowest probability that they will be reused. The assignment of these probabilities is described later. Contiguous sets are sought in order to avoid an additional pause to transmit commands required to compact the secondary display list. (2) When a contiguous set has not been found, the free blocks and the inactive DV's with lowest reuse probabilities are collected into a list, weights assigned, and a good set chosen to be deleted. The secondary display list subsequently must be compacted. The weight assigned to each DV is a function of its size, its position in the secondary display list (which determines the delay necessary to compact the list if this DV is deleted), and the reuse probability. The optimum set, that which provides enough space while minimizing the summed weights of DV's to be removed, can only be chosen while examining all possible combinations. Since this would burden the host processor, an alternative, approximate stepwise procedure is used. First, the list is sorted by weights and partitioned into two groups. Initially, the first group will contain the first *n* elements of the list whose total size is sufficient. The procedure then compares the sum of the weights of each pair of elements in the first group with the weight of an element of the second group. An element in the second group replaces the pair in the first when its weight is less and its size is greater than or equal to the corresponding pair in the first group. Similarly, the process is repeated by comparing combinations of three elements in the first group with pairs and single elements of the second group. We have found that continuing the search beyond three combinations does not yield a significant improvement. (3) If a check made prior to pursuing (1) and (2) reveals that the total free space and that occupied by inactive DV's is less than the space required, the operator is permitted to delete DV's from the active display by using the light-pen. This final means of obtaining the required space in the display buffer is provided primarily for program debugging, permitting the program to continue when it otherwise would have been abortcd.

Look-ahead facilities

The communications link between the graphics terminal and the host processor is normally idle for several seconds between each interaction. During this interval, the user is determining his next step. This suggests that if the host processor has a reasonable probability of predicting the user's choice, the effective transmission speed of the data link could be increased by using the idle time to transmit the graphics data most likely required for the next display frame. The display program can be represented as a discrete Markov chain whose states are all the possible display frames which the program may generate. In GRAF, the states are defined by the DV's which generate the display frame. The transition from one state to the next occurs at each interaction (light-pen detection, keyboard entry, or terminal processor interrupt).

To predict which state is most likely to occur next, we form a transition probability matrix

$$P_{ij}^{n,n+1} = P_r\{X_{n+1} = j \mid X_n = i\}$$

$P_{ij}^{n,n+1}$ is the probability that j will be the display program's next state given that i is the current state.

Each time the program enters a waiting state for some form of remote processor interrupt, the transition probabilities of the current state are examined and the DV's corresponding to the most probably next state of the program are transferred to the remote processor to the extent permitted by time and by space in the display buffer. Not all of the DV's corresponding to the next state can be transferred in advance since some may not be defined until after the operator interaction is completed and others, although fully specified, may not have been generated in advance.

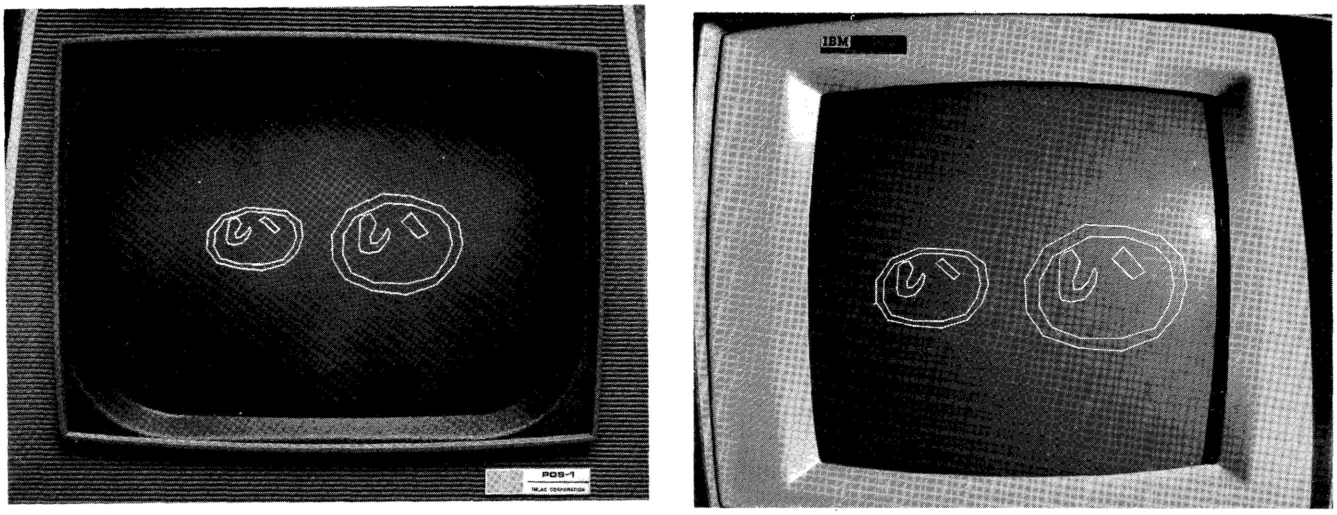
The state matrices are of two types: When the program is being executed for the first time, no information about the transition probabilities exists. All probabilities are assumed equal unless the programmer specifies otherwise. The second type of state matrix represents a typical use profile for the program which has been accumulated through many executions of the program. Even when an extensive use profiles exists, considerable improvements in the predictions can be made by dynamically updating the transition matrix so that it will more accurately predict the current use of the program. Provisions are made for each user to save and update his own transition matrix to tailor the program's response to his own needs. For example, new users of the intracavitary radiotherapy program will tend to seek more explanatory material and to elect more conservative approaches for locating the afterloaders. They will tend initially to use the transition matrix provided by the system, changing to their own when familiarity with the program motivates expedition of shortcuts and other preferred routings through the program. Thus, the system's matrix for each program will tend to remain oriented toward its most likely client, the new user.

Data-compression techniques

Processing capabilities of the graphics terminal enable unpacking of information transmitted from the host processor in a variety of condensed formats. While computations required for condensation can be complex without taxing a fast host machine, unpacking must be efficient enough to provide the desired advantage over unpacked transmission formats. The radiotherapy programs have directed attention to the problem of compacting information required for specifying graphical displays. Investigations proceeding on the 360/91 and 2250 while basic GRAF software is being tested are mindful of the two types of local graphics interfaces provided by the remote terminal: (1) specialized function generators such as those supporting curvilinear display generators, the light-pen tracking routine, routines which can translate, scale, and window display variables; and (2) routines which are written in the IMLAC PDS-1 machine code by applications programmers. The routines for both types of interface are assembled by the host processor and stored in a library to be dynamically loaded into the remote processor as they are required by the graphics programs. For purposes of memory management, the relocatable PDS-1 programs are considered to be display variables. Remote processor programs can extend the capability of the entire remote graphics terminal system, particularly where display dynamics are involved.

More general, conventional approaches to parameterized representation of graphical data are being explored initially, e.g., conic sections[19] and families of exponential curves,[20] as alternatives to piecewise representation by straight-line segments. The combined processing capabilities of this system permit additional approaches to be explored without the immediate constraint that they be implementable in hardware. For instance, for specification of organ contours or the distribution of fields around radioactive sources or X-ray beams, the approximation of large segments by French curves visually fitted at the terminal is being investigated.

Graphical information originating at the host processor, such as the dose distributions calculated for a given patient, cannot benefit initially from interactive approaches such as the latter. However, since considerable computation is required for each point in the dose-distribution field, procedures that transmit graphical



(a) IMLAC: (b) IBM 2250

Figure 6—Polar transform with different expansion moduli: Among approaches being explored to compact patient anatomical information for subsequent rapid transmission is representation of an individual's configuration by transforms mapping it into standard anatomical cross-sections. The outer contour on the left (non-anatomical) figure is being mapped into that shown on the right by expansions around a central point that assume different "expandability" of various domains. In this case, the outer (fat) layer is assumed to vary more than other "tissues."

information as the doses are being calculated alleviate much of the transmission delay. They have the additional advantage of enabling the radiotherapist to economize by terminating computations when the information displayed suffices for decision making. The contours shown in Figure 3 were generated line by line. The line at which computations are to commence may be indicated by the user. Another routine is available which computes and displays contour by contour.

An interest in transforms of related sets of graphical data is aroused by the awareness that a number of radiotherapists estimate internal-organ location by adjusting standard anatomical atlas cross-sections to measurements of the patient's external contour. We are exploring an improved version of this approach which allows different size-preserving factors for various tissues to be observed while transforms are being applied (see Figure 6). The objective is to reconstruct an adequate display of an individual's anatomical cross-section, using processing capabilities of the graphics terminal, locally stored standard cross-sections, and a small number of parameters characterizing a previously determined resultant transform. The latter would be obtained initially at the graphics terminal, as the radiotherapist seeks by a number of transforms on the entire cross-section or on individual organs to achieve an adequate correspondence between a transformed atlas cross-section and that determined for the patient by tomography or other methods.

By storage-management and look-ahead facilities, we have sought to provide users the most economical terminal possible. However, should radiotherapists wish to add a disk at the terminal for more complete anatomical atlas storage or capacity for additional stand-alone applications on the IMLAC, it would be worthwhile to investigate transferring to the terminal processor a stand-alone capability for many conversational portions of anatomical and treatment specification, to reduce subsequent communications charges. These program developments would be expedited by a 360/91 FORTRAN compiler that generates object code for the IMLAC.

Performance measurements

At this writing, we have not had enough experience with user programs to give any quantitative assessment of the remote graphics terminal's performance as compared to the 2250 terminal. There are areas where further development is already indicated. Decreasing response time by facilitating programmer's access to the remote computer through a compiler or by the addition of hardware components are areas to which we are now turning our attention.

In order to evaluate performance of the system and to quantitatively measure the effect of changes in the hardware and software, an extensive data collecting facility has been built into the programming packages. The programs continually log information regarding response time, data link utilization, storage utilization, display variable characteristics and other parameters which are believed to relate to system performance. This information is printed for the programmer at the end of each graphics job and is accumulated in a historical file for later analysis. Being provided with performance information regarding the execution of his graphics program, the user is likely to take an interest in those aspects of the program which affect its performance and to take the necessary steps to improve his program. The data gathering routines of the graphics system can also be instructed by an execution time parameter to collect a complete profile of a graphics job to the extent that its execution can be reproduced by using the profile as an input source instead of the terminal. This is particularly useful in debugging the system by providing a reproducible set of test jobs which can be run over and over until an error is found and corrected or the desired performance improvements have been made.

REFERENCES

1 C LEVINTHAL
Molecular model building by computer
Scientific American Vol 214 No 6 pp 42-52 1969

2 P W NEURATH B L BABLOUZIAN
T H WARMS et al
Human chromosome analysis by computer—An optical pattern recognition problem
Ann N Y Acad Sci Vol 128 pp 1013-1028 1966

3 H S FREY
An interactive program for chromosome analysis
Computers and Biomedical Research Vol 2 pp 274-290 1969

4 W F HOLMES
External beam treatment-planning with the programmed console
Radiology Vol 94 No 2 pp 391-400 1970

5 W J DIXON
Annual report 1968-1969
Health Sciences Computing Facility University of California Los Angeles Appendix B and pp 83-85

6 T L LINCOLN
The clinical significance of simulation and modeling in leukemia chemotherapy
Proceedings of the Spring Joint Computer Conference 1972 In Press

7 P M BRITT W J DIXON R I JENNRICH
Time-sharing and interactive statistics
Bull ISI Proceedings of the 37-th Session pp 191-207 1969

8 *User's guide for GRAF: Graphics additions to FORTRAN*
Health Sciences Computing Facility University of California Los Angeles 1968

9 A HURWITZ J P CITRON J B YEATON
GRAF: Graphic additions to FORTRAN
Proceedings of the Spring Joint Computer Conference Vol 30 pp 553-557 1967

10 R C UZGALIS H FREY
PL/OT, Graphics Subroutines for PL/1
Health Sciences Computing Facility University of California Los Angeles 1969

11 C M NEWTON
Remote graphics radiotherapy treatment planning
Proceedings USPHS-Mt. Sinai Afterloading Conference New York May 6-8 1971 In Press

12 M STOVALL
Bibliography of computers in radiation therapy
Proceedings of the Third International Conference on Computers in Radiation Therapy Glasgow September 1970 Special Edition British Journal of Radiology In Press

13 G K BAHR J G KEREIAKES H HORWITZ
R FINNEY J GALVIN K GOODE
The method of linear programming applied to radiation treatment planning
Radiology Vol 91 pp 686-697 1968

14 C S HOPE J S ORR
Computer optimization of 4-Mev treatment planning
Phys Med Biol Vol 10 pp 365-373 1965

15 C M NEWTON
What next in radiation treatment optimization?
Proceedings of the Third International Conference on Computers in Radiation Therapy Glasgow September 1970 Special Edition British Journal of Radiology In Press

16 C CHRISTENSEN E N PINSON
Multi-function graphics for a large computer system
Proceedings of the Fall Joint Computer Conference Vol 31 pp 697-711 1967

17 I W COTTON F S GREATOREX
Data structures and techniques for remote computer graphics
Proceedings of the Fall Joint Computer Conference Vol 33 pp 533-544 1968

18 M D RAPKIN O M ABU-GHEIDA
Stand-alone/remote graphic system
Proceedings of the Fall Joint Computer Conference Vol 33 pp 731-746 1968

19 L G ROBERTS
Conic display generator using multiplying digital-analog converters
IEEE Transactions on Electronic Computers Vol EC-16 No 3 pp 369-370 1967

20 M L DERTOUZOS
Phaseplot: an online graphical display technique
IEEE Transactions on Electronic Computers Vol EC-16 No 2 pp 203-209 1967

Automated information-handling in pharmacology research

by WILLIAM F. RAUB

National Institute of Health
Bethesda, Maryland

Pharmacology in its broadest sense involves the multitude of interrelationships between chemical substances and the function of living systems. Since these interrelationships manifest themselves at all levels of physiological organization from the individual enzyme to the intact mammal, research in this area involves concepts and techniques from almost every biomedical discipline. Detailed understanding of the mechanisms of drug action is the best prescription for the discovery of new therapeutic agents and the rational use of old ones.

The breadth and complexity of the domain of pharmacological inquiry interact to produce a class of information-handling problems as formidable and enticing as any that can be found in the medical area. In recognition of this, the National Institutes of Health (NIH), through its Chemical/Biological Information-Handling (CBIH) Program, is attempting to accelerate the acquisition of new pharmacological knowledge by designing and developing computer-based research tools especially for scientists in this discipline. Working through a tightly interconnected set of contracts with universities and colleges, nonprofit research institutes, profit-making organizations, and Government agencies, the CBIH Program seeks to blend the most advanced information science methods into a computer system which can be an almost indispensable logistical and cognitive aid to these investigators. This paper characterizes the current status of these efforts.

THE PHARMACOLOGICAL INFORMATION-HANDLING "PROBLEM"

The ultimate goal of pharmacology is a well-validated theory having the following two properties:

(1) given the identity of a chemical substance and the parameters of its administration to a given living organism (e.g., the dose, route, course, etc.), predict the effect, if any, on the behavior of the biological system; and

(2) given a desired behavioral end state for a given biological system, predict the specific chemical substance(s) and parameters of administration which can bring about that end state.

Only when such a theory is in hand will it be possible to make an accurate assessment of the therapeutic efficacy of a new substance without extensive testing in animals and human subjects. Only then will one be able to assume with confidence that a given drug will not have adverse side effects in a particular clinical context.

As is painfully obvious to both pharmacologists and laymen alike, an all-encompassing theory of drug action is nowhere in sight. Present understanding of drug action still is based almost exclusively upon empirical observation. Only a figurative handful of investigators have made extensive attempts to formalize their knowledge through mathematical modelling and the like. Natural language still serves as the predominant medium for the expression and communication of concepts. And textbooks and review articles constitute almost the entire body of organized knowledge.

Nor has information science and technology done much to aid in the development of a broad-based theory of drug action. On the contrary, even the most sophisticated document reference and archival management systems are only marginally useful in helping pharmacologists cope with the plethora of literature which is relevant to their interests. Moreover, outside of a few parochial areas, there are essentially no truly effective data retrieval (as opposed to document retrieval) services. And easy-to-use tools for building, exercising, managing, and communicating models of drug action are much more nearly fantasy than they are routine offerings of the typical general-purpose computer center.

Thus, the pharmacological information-handling

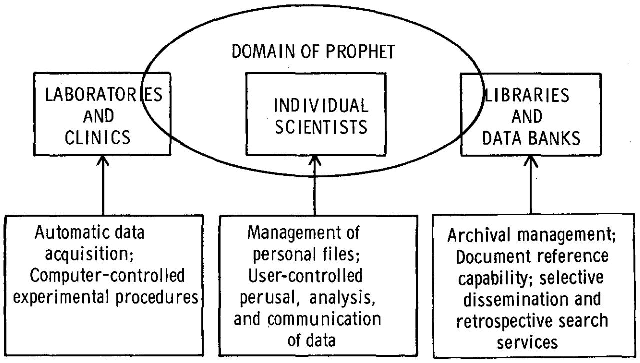

DOMAIN OF PROPHET
LABORATORIES AND CLINICS
INDIVIDUAL SCIENTISTS
LIBRARIES AND DATA BANKS
Automatic data acquisition; Computer-controlled experimental procedures
Management of personal files; User-controlled perusal, analysis, and communication of data
Archival management; Document reference capability; selective dissemination and retrospective search services

Figure 1—The PROPHET system in context

"problem" can be stated as a series of interrelated questions:

(1) Can one design computer-based information-handling tools which encourage formalistic rather than empirical approaches to the study of molecular structure/biological activity relationships?
(2) Can one use these tools to facilitate the interchange of data, procedures, and models among geographically and disciplinarily disjoint scientists whose work is relevant to the understanding of drug action?
(3) Can one use mathematical models and other formalisms to enhance traditional methods for the storage and retrieval of pharmacological information?

These questions circumscribe the domain of the CBIH Program. Some very preliminary answers are included in later sections of this discussion.

RELATED WORK

The interests of the CBIH Program in attacking the pharmacological information-handling "problem" intersect in one way or another with a large number of activities in the information science area, for the rubric "computers in pharmacology" covers a wealth of topics. As depicted in Figure 1, the CBIH Program in its work on the PROPHET System and related projects focuses on only a small—albeit crucial—subset of the potentially valuable applications for automated information-handling methods. Most notable is the absence of concern with (a) data acquisition and process control for experimental laboratory and clinical care settings and (b) automated archival management methods and other uses of computers in libraries. Since these topics are only of marginal interest to the CBIH Program (though obviously of utmost importance from other perspectives), they are not treated further in this account.

Turning attention specifically to the topic of personalized pharmacological data-management and analysis (i.e., the middle portion of the spectrum shown in Figure 1), there are two major classes of activity upon which the CBIH Program draws. The first is the collection of efforts concerned with chemical information-handling, especially those involving highly dynamic man/machine interaction about molecular structure data. The second is the collection of efforts concerned with general-purpose data management, especially those systems which offer the user great flexibility in the definition of file structure and which are designed to operate in a time-sharing environment.

Chemical information

The use of computers in handling chemical data (primarily structural information) is widespread and dates back to the mid-1950's. The need to deal most effectively with large numbers of organic molecules (in some cases hundreds of thousands) provides the continuing strong motivation; the fact that structural data is readily codified for machine manipulation makes this aspect of chemical information processing the most nearly tractable—and, hence, the most popular. Among the organizations who have made significant contributions are the Chemical Abstracts Services, the Walter Reed Army Medical Center, the Institute for Science Information, and a number of leading chemical and pharmaceutical manufacturers. The bulk of the efforts in this field are reviewed in two publications of the National Academy of Sciences-National Research Council.[1,2]

The subject of greatest concern in the chemical information field over the years has been the selection of an appropriate notation for encoding molecular topology. Almost every group has struggled with the choice between connectivity tables and linear ciphers. The connectivity tables have the advantage of being direct atom-by-atom/bond-by-bond descriptions of molecules and, hence, they allow one to perform searches for essentially any subgraph without the need for predetermined indices. Connectivity tables invariably make heavy demands upon storage, however, and, in the case of even medium-sized files (e.g., 10,000 structure records) the extensive processing time required for many substructure searches renders them impractical for production use without some fairly rigorous preliminary screening of the queries to the file. By contrast, linear notations are compact in terms of internal representation, are processed with relative ease and economy, and, with some perseverance, can be mastered by the scientist for direct use as a form of nomenclature. But linear notations do not allow readily for the full range of possible substructure queries, and this shortcoming has proved serious in some cases.

Obviously, the strengths and weaknesses of these two basic approaches are highly context-dependent, and there is not likely to be a move toward consensus until additional advances in retrieval methods (e.g., in hash-coding strategies) are made. Moreover, since there are many possible compromises (e.g., the linear notations are useful in substructure searching as screening terms preliminary to full-scale topological tracing), the study of search and retrieval methods for chemical information is almost certain to be an active area for the foreseeable future. And since chemical structure data is of direct concern to pharmacologists, the CBIH Program will attempt to insure that the pertinent results of the many efforts involving new chemical information-handling methods will be available for their use.

Another major problem area in the chemical information field is the man/machine interface, and in this case CBIH Porgram contractors have played a rather substantial role. Especially noteworthy are (a) the efforts of Corey and Wipke[3] on easy input of two-dimensional chemical graphs via computer-controlled stylus and tablet and (b) the efforts of Levinthal and Katz[4] on the computer-driven display and manipulation of three-dimensional molecular models. Both of these groups have amply demonstrated the viability of these man/machine methods in the context of medically relevant research—Corey and Wipke in the area of artificially intelligent systems for the design of organic chemical syntheses and Levinthal and Katz in the area of macromolecule structure and function. Moreover, as described further below, a number of the results of these groups have been integrated by yet a third CBIH Program contractor (Bolt Beranek and Newman Inc.) into a powerful computer graphics front end for a pharmacological information-handling system called PROPHET.

General-purpose data management

In parallel with the work on chemical information-handling via computer, the concept of a general-purpose data management system has received considerable attention. In brief, many groups have recognized the need for a software system which can be used to manage a wide range of files flexibly and efficiently and with little, if any, requirement for programming on the part of the user. Among the most active organizations are System Development Corporation, Computer Corporation of America, and many of the major equipment vendors. A somewhat dated but still useful characterization of work in this area is given in the review article of Minker and Sable.[5]

From the CBIH Program perspective, the most interesting data management systems are those designed to operate in a time-sharing environment. Since personalized information-handling tools for pharmacologists almost inevitably will need be highly interactive if they are to be effective, the successes and shortcomings of the many ongoing efforts involving time-shared data management systems will do much to influence how this technology is applied in support of drug research. For example, the various compromises involving the degree of file inversion (in part, a trade-off between the speed of file update and the speed of retrieval) and the degree of optimization to a particular storage medium (in part, a trade-off between speed and generality) are especially instructive. Moreover, within the CBIH Program's own activities, the use of a powerful interactive text editor as a simple file management tool for the study of structure/activity relationships in morphine-like substances has served to clarify a number of system design questions.[6]

MAJOR CBIH PROGRAM EFFORTS

Turning to the currently ongoing activities receiving direct CBIH Program sponsorship, the remainder of this discussion is devoted to two projects: (a) the design and prototype implementation of the PROPHET System by Castleman, Briscoe, and colleagues at Bolt Beranek and Newman Inc. (BBN) and (b) the studies of automated management of pharmacological knowledge by Werner, Pople, and associates at the University of Pittsburgh. Both of these projects are extensive, multiyear endeavors and can only be described in broad outline here; nor do they constitute the total range of current program efforts. Nevertheless, they best portray the CBIH Program's strategy for dealing with the pharmacological information-handling "problem," and their early results provide invaluable indicators of the promises and difficulties which lie ahead.

The PROPHET system

PROPHET is an experimental information-handling system designed specifically to subserve pharmacology

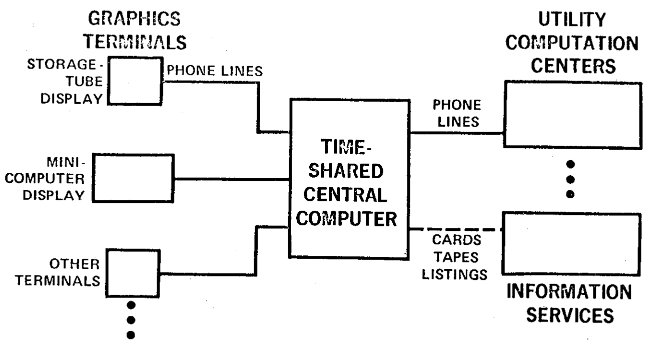


Figure 2—PROPHET system equipment organization

research. It is intended to be a medium through which the latest pharmacologically relevant information-handling methods can be developed, integrated, and made available to practicing scientists throughout the nation. It is viewed as a research tool encouraging more rigorous approaches to the organization and analysis of laboratory and clinical observations. Its ultimate goal is to facilitate the acquisition and dissemination of knowledge about mechanisms of drug action across a span of disciplines ranging from molecular biology to human clinical investigation.

The general nature of PROPHET is given in Figure 2. As shown there, PROPHET is designed to employ a dedicated, medium-sized, time-sharing computer (a Digital Equipment Corporation PDP-10) and to serve a geographically dispersed set of users via remote graphics terminals operating over voice-grade telephone lines. Information-handling procedures and data files reside in the central computer. Provision is made for eventual interfacing with both utility computers and information services. A detailed description of the system is given in the document entitled "The PROPHET System: A Final Report of Phase I of the Design Effort for the Chemical/Biological Information-Handling Program."[7]

The primary features of PROPHET as seen from the perspective of computer technology are five:

(1) a powerful interactive command language enabling even a computer novice to effect sophisticated procedures for handling empirical data on chemical substances and their biological consequences;

(2) a simple procedural language syntactically modelled on PL-1 which is coextensive with the command language and which allows essentially unfettered interleaving of system and user-defined functions;

(3) a special emphasis on making complex computational processes relatively invisible to the pharmacologist user (e.g., presenting sophisticated data management strictly in terms of operations on tables, allowing stylus and tablet input and CRT display of molecular structures, etc.);

(4) a rich substrate of special data types for handling such pharmacologically relevant data structures as tables, molecules, graphs, and nodal networks; and

(5) a design which attempts to circumscribe the full power of a general-purpose digital computer in such a way that communication among users is strongly facilitated without inappropriately constraining options for individual initiative.

In other words, by specifically encouraging user-system and user-user interactions within the somewhat rigid context of standard predetermined data types, PROPHET is designed to allow research pharmacologists to handle their personal data files and communicate with their colleagues more effectively than they can any other way.

Although PROPHET, like most complex computer systems, is better demonstrated than described, a few simple illustrations may prove useful. Figures 3 and 4 show the system commands for operating on tables and molecules, respectively. Arguments for the commands are given in small letters; square brackets enclose optional portions; braces enclose alternative forms. These two lists are only a fraction of the commands available to the user. Note especially the integration of the various data types—e.g., the data type molecule can be an entry in the cell of a table.

```
1. MAKE TABLE unused-name [FROM table-name [FOR relational expression]]

2. ADD {COLUMN[S]} TO table-name [FROM table-name]
       {ROW[S]   }

3. ENTER [ROW[S] {row-name(s)  }              ] OF table-name
         [       {row-number(s)}              ]
         [COLUMN[S] {column-name(s)  }        ]
         [          {column-number(s)}        ]
4. FILLIN table-name FROM table-name

5. {DISPLAY} [[COLUMN[S]  {column-name(s)  } OF]]
   { PRINT }             {column-number(s)}
             [[ROW[S] {row-name(s)  } OF]   table-name]
                      {row-number(s)}
6. LIST{OTHER                  } TABLES
       {log-in identifiicaton }

7. SORT table-name BY COLUMN[S] {column-name(s)  }
                                {column-number(s)}

8. PROTECT table-name
```

Figure 3—Summary of table-handling commands

1. MAKE MOLECULE unused-name [FROM display-indication]
2. DISPLAY [CONFORMATION OF] molecule-name [COMPLETE]
3. COMPUTE MODEL OF molecule-name
4. CHEMSET property-name TO value(s) { element-indication(s)
 number(s) }
5. WHAT IS property-name { element-indication(s)
 number(s) }
6. ADD COLUMN[S] TO { BONDS
 ATOMS } of molecule-name
7. [UN] LABEL property-name of molecule-name
8. ROTATE [DISPLAY] BOND number { element-indications
 numbers }
9. DISPLAY molecule-name WITH molecule-fragment-name

Figure 4—Summary of molecule-handling commands

The nature of PL/PROPHET, the procedural language is illustrated in Figures 5-8 using data drawn from a study of antileukemic drugs being conducted by the Southern Research Institute in Birmingham, Alabama. Figures 5 and 6 show selected columns from a user-defined table set up to handle survival time data on mice inoculated with various quantities of L-1210 leukemic cells; the numbers in the cells of Figure 6 indicate the number of mice which died on that particular day. Note that columns 19 and 20 of the table (Figure 5) are blank since they are intended to accommodate results derived from earlier columns.

Figure 7 lists a brief user-defined procedure for calculating the number of 11-day survivors and the mean survival time from the observations recorded in columns 4-14 and storing them in columns 19 and 20. Figure 8 then gives the result of calling this new function—i.e., an updated version of the table with the derived values in place. Note that the user need take no special action to have his personal PL/PROPHET procedures operate in conjunction with the standard system functions.

*DISPLAY COLUMNS 1,2,3,19,20 OF MEGAMOUSE$ *

MEGAMOUSE 8R X 20C	1. IMPLANT CELLS	2. DOSAGE (MG/KG)	3. SCHEDULE	19. 11 DAY SURVIVORS /TOTAL	20. MEAN LIFE SPAN
1	1.E7	2.25E2	Q7D,DAYS		
2	1.E7	1.5E2	2,9,16		
3	1.E7	1.E2			
4	1.E7	6.7E1			
5	1.E6	2.25E2	Q7D,DAYS		
6	1.E6	1.5E2	2,9,16		
7	1.E6	1.E2			
8	1.E6	6.7E1			

Figure 5a—Mouse survival time: Selected columns of the table "Megamouse"

*DISPLAY COLUMNS 4-14 OF MEGAMOUSE$

MEGAMOUSE 8R X 20C	4. DAY 4	5. DAY 5	6. DAY 6	7. DAY 7	8. DAY 8	9. DAY 9	10. DAY 10	11. DAY 11	12. DAY 12	13. DAY 13	14. DAY 14
1							2		1	1	1
2								1		1	
3						1					1
4				2	3	4					
5	1		1					1	1		
6											
7									1	1	1
8				1	1		3	2		1	2

Figure 5b—Mouse survival time: Selected columns of the table "Megamouse"

```
SURVIVE: PROCEDURE (TABNAM);

DECLARE (I,J,DEATH,SUMDEATH,SURVS) FIXED;
DECLARE TABNAM TABLE;
DECLARE (ENTRYIJ,ENTRY19,NU,TOT) CHARACTER;

NU = ' ';
TOT = '/10';

DO I = 1 TO 8;
  SUMDEATH = 0;
  DEATHDAYS = 0.0;
  DO J = 4 TO 14;
    SET ENTRYIJ TO COLUMN J ROW I OF TABNAM;
    IF ENTRYIJ=NU THEN GO TO ABC;
    DEATH = CINT(ENTRYIJ);
    SUMDEATH = SUMDEATH + DEATH;
    DEATHDAYS = DEATHDAYS + CREAL(DEATH)*CREAL(J);
  ABC: END;
  SURVS = 10 – SUMDEATH;
  ENTRY 19 = CSTRING(SURVS).TOT;
  ENTRY20 = DEATHDAYS/CREAL(SUMDEATH);
  SET COLUMN 19 ROW I OF TABNAM TO ENTRY19;
  SET COLUMN 20 ROW I OF TABNAM TO ENTRY20;
END;

RETURN;
END;
```

Figure 6—Listing of PL/PROPHET procedure "Survive" to analyze mouse survival time data

*CALL SURVIVE (MEGAMOUSE)$ *ERASE ALL$ *DISPLAY COLUMNS 2,3,19,20 OF MEGAMOUSE$

MEGAMOUSE 8R X 20C	2. DOSAGE (MG/KG)	3. SCHEDULE	19. 110DAY SURVIVORS /TOTAL	20. MEAN LIFE SPAN
1	2.25E2	Q7D,DAYS	3/10	1.3E1
2	1.5E2	2,9,16	5/10	1.46E1
3	1.E2		4/10	1.466666E1
4	6.7E1		0/10	9.1
5	2.25E2	Q7D,DAYS	4/10	1.1E1
6	1.5E2	2,9,16	7/10	1.7E1
7	1.E2		5/10	1.48E1
8	6.7E1		0/10	1.079999E1

Figure 7—Selected columns of the table "Megamouse" showing results of the procedure "Survive"

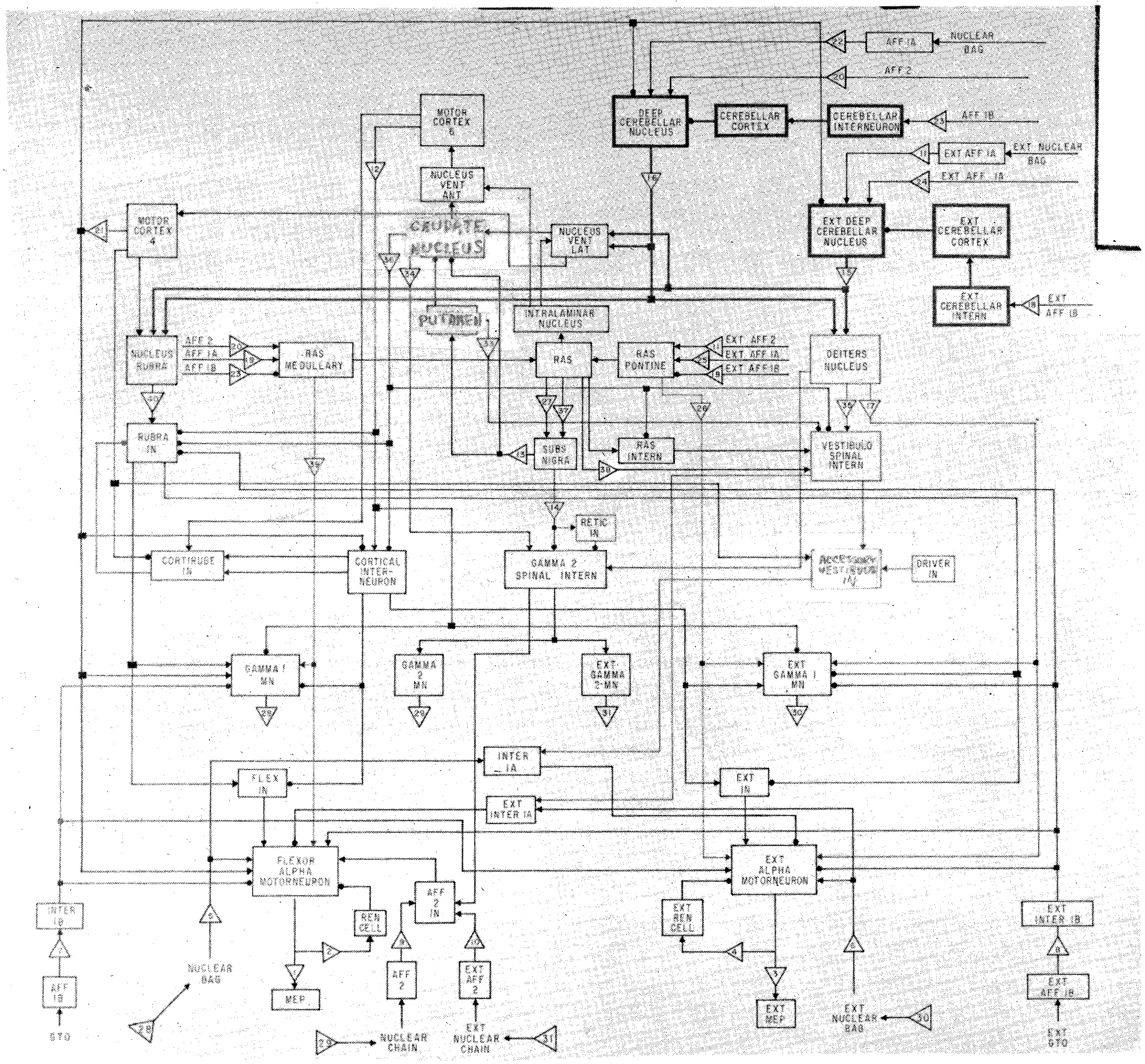


Figure 8—Simple model of neural control of skeletal muscle movement

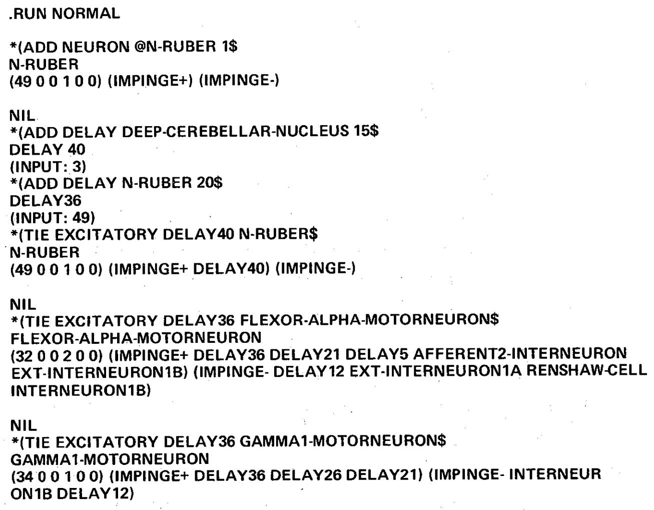
At the present time PROPHET is available on a dedicated PDP-10 computer housed and operated for the CBIH Program by First Data Corporation of Waltham, Massachusetts, and is ready for extensive evaluation by research pharmacologists in the context of their day-to-day activities. It is hoped that the testing period now beginning will help both to refine current concepts and to identify areas for future investigation. It is also hoped that these collaborative projects using PROPHET will result in clear-cut demonstrations of the utility of modern information-handling methods in the search to understand the mechanisms of drug action.

Automated management of pharmacological knowledge

A vital complement to the initial work on the PROPHET System is the basic research being conducted by Werner, Pople, and their associates at the University of Pittsburgh.[8] This group is exploring the problems and promises of using automated inference methods as cognitive aids to the individual pharmacological investigator. They seek to demonstrate where and how the concepts of artificial intelligence might be applied to achieve marked qualitative and quantitative improvements in current capabilities for the storage and retrieval of information.

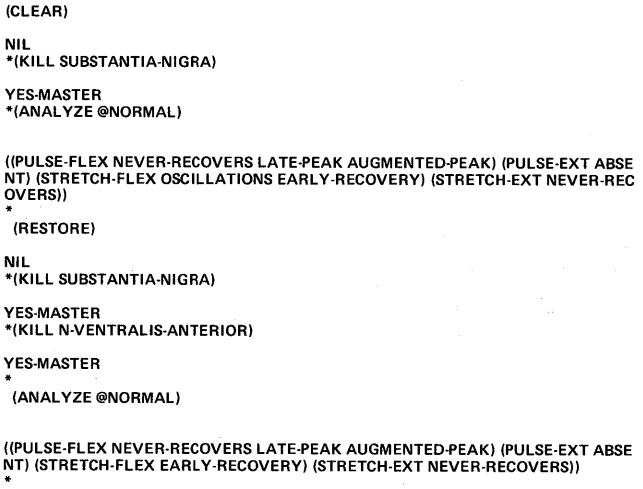
The Pittsburgh team has concentrated the bulk of its efforts to date on model-handling tools—i.e., system functions which make it relatively easy for even the computer novice to construct and exercise models of pharmacological processes, assuming, of course, that he has the basic biomedical knowledge to engage realistically in such an endeavor. The intent is to aid individual investigators not only in expressing their concepts about mechanisms of drug action but also in assessing the validity of those conceptualizations through digital simulation and other algorithmic processes. Put another way, the goal is to produce an easy-to-use software system whose capacity to absorb and operate upon pharmacological knowledge is unprecedented in its scope and power.

Werner, Pople, and their associates have found especially useful the class of models known as finite state automata. In brief, the builder of the model specifies each node in a network in terms of algorithmic processes, direction and nature of connectivity, threshold logic and other control information, and the like. Proceeding a "fact" at a time, so to speak, the user can build arbitrarily large and complex model structures. To effect a simulation, the state of each node in the network is determined at each of a succession of discrete time steps using the algorithms and other information provided at the time of model construction. Moreover, by having the user (or another computer program) monitor the time course of the state transitions at selected nodes, one can perceive the behavior of the model in response to various stresses and compare this behavior with observations made in the laboratory or clinic.



```
.RUN NORMAL

*(ADD NEURON @N-RUBER 1$
N-RUBER
(49 0 0 1 0 0) (IMPINGE+) (IMPINGE-)

NIL
*(ADD DELAY DEEP-CEREBELLAR-NUCLEUS 15$
DELAY 40
(INPUT: 3)
*(ADD DELAY N-RUBER 20$
DELAY36
(INPUT: 49)
*(TIE EXCITATORY DELAY40 N-RUBER$
N-RUBER
(49 0 0 1 0 0) (IMPINGE+ DELAY40) (IMPINGE-)

NIL
*(TIE EXCITATORY DELAY36 FLEXOR-ALPHA-MOTORNEURON$
FLEXOR-ALPHA-MOTORNEURON
(32 0 0 2 0 0) (IMPINGE+ DELAY36 DELAY21 DELAY5 AFFERENT2-INTERNEURON
EXT-INTERNEURON1B) (IMPINGE- DELAY12 EXT-INTERNEURON1A RENSHAW-CELL
INTERNEURON1B)

NIL
*(TIE EXCITATORY DELAY36 GAMMA1-MOTORNEURON$
GAMMA1-MOTORNEURON
(34 0 0 1 0 0) (IMPINGE+ DELAY36 DELAY26 DELAY21) (IMPINGE- INTERNEUR
ON1B DELAY12)
```

Figure 9—Use of the model building commands "Add" and "Tie"

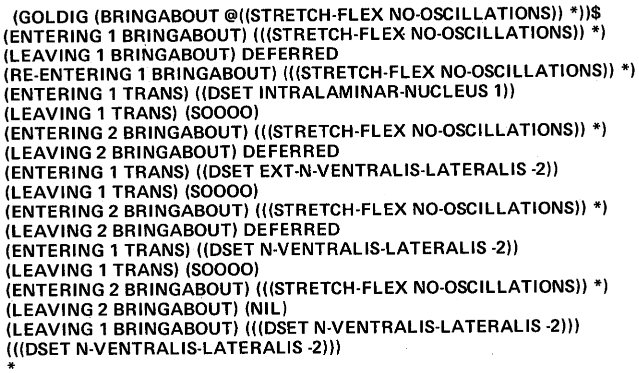


```
(CLEAR)

NIL
*(KILL SUBSTANTIA-NIGRA)

YES-MASTER
*(ANALYZE @NORMAL)


((PULSE-FLEX NEVER-RECOVERS LATE-PEAK AUGMENTED-PEAK) (PULSE-EXT ABSE
NT) (STRETCH-FLEX OSCILLATIONS EARLY-RECOVERY) (STRETCH-EXT NEVER-REC
OVERS))
*
 (RESTORE)

NIL
*(KILL SUBSTANTIA-NIGRA)

YES-MASTER
*(KILL N-VENTRALIS-ANTERIOR)

YES-MASTER
*
 (ANALYZE @NORMAL)


((PULSE-FLEX NEVER-RECOVERS LATE-PEAK AUGMENTED-PEAK) (PULSE-EXT ABSE
NT) (STRETCH-FLEX EARLY-RECOVERY) (STRETCH-EXT NEVER-RECOVERS))
*
```

Figure 10—Use of the model exercising command "Analyze"

The neuropharmacology associated with the control of skeletal muscle movement has served as the prime subject for the Pittsburgh group's early efforts. As shown by Figure 8, even a simple model of this pharmacological system is essentially unmanageable without computer assistance. However, using an early version of their model-handling tools, these investigators have achieved some most encouraging results. While the following brief series of examples can hardly given an adequate characterization, they should impart the general flavor of the work.

Figure 9 shows how the model building commands (i.e., ADD, DELETE, CUT and TIE) are used; in this case the user is adding the Nucleus Ruber to the model, assigning it a threshold value of 1 and connecting it via appropriate delays so that it gets excitatory input from the Deep Cerebellar Nucleus (already in the model) and delivers excitatory output to the alpha- and gamma-1 motorneurons on the flexor side. Figure 10 illustrates the model exercising and perception functions; in this case the command ANALYZE is entered, causing the model to be stressed by a standard set of stimuli analogous to the tendon taps and passive muscle stretch used routinely in clinical neurology examinations. The user has the option of setting a window



```
 (GOLDIG (BRINGABOUT @((STRETCH-FLEX NO-OSCILLATIONS)) *))$
(ENTERING 1 BRINGABOUT) (((STRETCH-FLEX NO-OSCILLATIONS)) *)
(LEAVING 1 BRINGABOUT) DEFERRED
(RE-ENTERING 1 BRINGABOUT) (((STRETCH-FLEX NO-OSCILLATIONS)) *)
(ENTERING 1 TRANS) ((DSET INTRALAMINAR-NUCLEUS 1))
(LEAVING 1 TRANS) (SOOOO)
(ENTERING 2 BRINGABOUT) (((STRETCH-FLEX NO-OSCILLATIONS)) *)
(LEAVING 2 BRINGABOUT) DEFERRED
(ENTERING 1 TRANS) ((DSET EXT-N-VENTRALIS-LATERALIS -2))
(LEAVING 1 TRANS) (SOOOO)
(ENTERING 2 BRINGABOUT) (((STRETCH-FLEX NO-OSCILLATIONS)) *)
(LEAVING 2 BRINGABOUT) DEFERRED
(ENTERING 1 TRANS) ((DSET N-VENTRALIS-LATERALIS -2))
(LEAVING 1 TRANS) (SOOOO)
(ENTERING 2 BRINGABOUT) (((STRETCH-FLEX NO-OSCILLATIONS)) *)
(LEAVING 2 BRINGABOUT) (NIL)
(LEAVING 1 BRINGABOUT) (((DSET N-VENTRALIS-LATERALIS -2)))
(((DSET N-VENTRALIS-LATERALIS -2)))
*
```

Figure 11—Use of the model exercising command "Bringabout"

such that he may observe the time course of activity at only selected nodes. The command ANALYZE also invokes a perception program which monitors the time course of the simulated muscle length and, by applying a succession of linguistic abstractions to the output, produces a description of the model's behavior in clinical terminology.

Drug effects, of course, may also be studied with this system. In brief, the computer is provided with a list of drug names and the associated changes the drugs are thought to induce in the model's structure and/or control parameters. When the command to administer a given drug is entered, the software system automatically makes the indicated alterations. Note that surgical procedures (e.g., decerebration) can be handled via the same mechanism.

Emphasis also has been directed toward having other computer programs exercise the neural model. Pople has experimented with a problem solving system whereby a high level executive processor accepts user queries and answers them by an appropriate combination of structural modifications and simulation runs. This allows the user to investigate hypotheses about drug action on-line by observing how the system can BRINGABOUT specified end states in terms of structural and functional changes in the model such as drugs are known to cause. Figure 11 gives a brief example of some preliminary results. In this case the system is asked to find an alteration which eliminates the prolonged oscillations following muscle stretch on a model mimicking Parkinson's disease. After two unsatisfactory maneuvers (i.e., adding driving forces to the Intralaminar Nucleus and the External Nucleus Ventralis Lateralis, respectively), the BRINGABOUT command discovers that applying a driving force of -2 to the Nucleus Ventralis Lateralis leads to the goal state.

For the immediate future, the Pittsburgh group plans to continue its investigations into techniques whereby artificially intelligent computer programs can exercise or otherwise manipulate pharmacological models. For example, studies are now under way on such topics as (a) automatic selection of the appropriate level of resolution of a model when it is used in a specific question-answering context, (b) definition of a quasi-English language whose utterances are equivalent to data structures in the model, and (c) development of techniques for using a model to organize and retrieve references to the experimental literature which is associated with that model. These investigators have only scratched the surface on a fascinating and formidable collection of information-handling problems.

FUTURE PROSPECTS

The CBIH Program's progress to date represents a collection of small (but nevertheless real) inroads into an enormously difficult task. In the years ahead the Program must deal with three basic questions:

(a) How can the evolving PROPHET System best be integrated into the mainstream of pharmacological research?
(b) When and how can PROPHET be enriched with capabilities for model management?
(c) What additional research topics in the information science field should be addressed if the information storage and retrieval aspects of PROPHET are to be made most effective?

The first two of these questions will be met initially at least by continuing the work at BBN and the University of Pittsburgh, taking steps specifically to coordinate these efforts even more closely with one another and with leading pharmacology laboratories throughout the nation. The third question, to the extent that funds permit, will be approached by experimenting with new pharmacological data types and representation conventions and by exploring such notions as the utility of data description schema.

To recapitulate the opening theme, pharmacology offers an enormously rich problem space for those interested in applying computers in the biomedical world. Pharmacology impinges on almost every area of medical science and its progress is heavily dependent on the continuing erosion of disciplinary boundaries. Moreover, its information-handling requirements strain at the frontiers of computer science in almost every direction. And the overwhelmingly important implications for society that accompany even small advances in our understanding of drug action make it urgent that the challenge for further inquiry be accepted.

REFERENCES

1 M HUNSBERGER et al
Survey of chemical notation systems
National Academy of Sciences-National Research Council
Washington DC Publication 1150 1964

2 *Chemical structure information handling: A review of the literature 1962-1968*
National Academy of Sciences-National Research Council
Washington DC 1969

3 E J COREY W T WIPKE
Computer-assisted design of complex organic syntheses
Science 166: 178-92 1969
E J COREY et al
Progress reports
NIH Contract #PH-43-68-1018 NIH-70-4105

4 C LEVINTHAL et al
Progress reports
NIH Contracts #PH-43-67-1131 and #PH-43-68-1019
NIH-71-4028

5 J MINKER J SABLE
File organization and data management in annual review of information science and technology
Carlos A. Cuatra Ed American Documentation Institute
Annual Review Series Vol 2 Chap 7 Wiley Interscience
New York NY 1967

6 S J KAUFMANN et al
Computer-assisted data management in medicinal chemistry—The use of a general purpose text editor
J Chem Doc 10: 248-255 1970

7 P A CASTLEMAN et al
The PROPHET system: A final report on phase I of the design effort for the chemical/biological information handling program
NIH Contract #PH-43-68-1328
NIH-70-4113

8 G WERNER et al
Progress reports
NIH Contract #NIH-69-2073 NIH-70-4121

Where do we stand in implementing information systems?

by JAMES C. EMERY

University of Pennsylvania
Philadelphia, Pennsylvania

ACHIEVEMENTS OF THE COMPUTER INDUSTRY

By any reasonable yardstick the computer has been a fantastic success. It has grown in one generation from a laboratory curiosity to one of our major industries—perhaps the most vital one in our information-oriented post-industrial society. Raw computational speed has increased during this period by an order of magnitude every four or five years. The cost of computation has dropped to the point that we can employ it lavishly on everyday tasks.

The computer has already brought significant changes in science, engineering, and commerce. Some branches of technology, such as space exploration, would scarcely be possible without the computer. Virtually all scientific fields have been affected in varying degrees. Without the modern computer it would be very difficult to perform the myriad data processing tasks required to run a large organization. Indeed, many of the companies most heavily involved in data processing, such as those in the banking and insurance industries, could simply not exist in their present form without the computer.

PROBLEM IN APPLYING COMPUTERS

Despite these unprecedented achievements, we computer professionals are not universally hailed as heroes. In fact, we have a difficult time staving off our detractors. It would be comforting to think of them as ungrateful wretches who resent the bounty that we have bestowed upon them. Unfortunately, much of our difficulty stems from our own failures to exploit the technology nearly as well as we might. It is all too easy to identify these failures.

Underestimating complexity

Man seems to have a natural bias toward underestimating the complexity of information systems. It is easy to see how this can come about. When we think about a problem we tend to concentrate on the normal combination of circumstances; we often overlook the exceptions. And yet it is the exception that causes much of the work in designing and implementing an information system. It is not uncommon, for example, for over half of the design effort to go into the handling of exceptions that account for a few percent of the transactions entering the system. One approach, of course, is to handle the rare exception outside of the automatic system. But even this requires considerable design effort to detect the exception and control the eventual entry of the corresponding correction.

Just as man seems to have a proclivity to ignore the exception, he also seems to have a bias toward optimism. Estimates are frequently based on the assumption that everything will work out as planned, ignoring the overwhelming evidence to the contrary. Critical difficulties often appear in the process of linking individual subsystems into the overall system. System integration and testing typically account for over half of the total cost of implementing a complex system, but it is the rare estimate that provides this level of funding.

Reinforcing these psychological biases are certain institutional biases toward optimistic estimates. The cost of implementing a large system is very great, but many organizations have not been willing to face up to this fact. More than once I have heard an MIS director say that he could never get authorization approved if he gave honest estimates.

Underestimation of the complexity of a system leads to a number of unfortunate consequences. An obvious one is the cost and schedule overrun that we so often experience. This, in turn, often leads to hasty shortcuts that result in systems that are error-prone, inflexible, and poorly documented.

Design of overly sophisticated systems

Underestimating complexity can lead into the trap of striving for too much technical sophistication. A technician naturally wants to design systems that move to—or even extend—the frontier of technology. Often overlooked is the cost-effectiveness of marginally useful sophistication.

A common example is an overemphasis on quick response. On-line file updating and retrieval, with response times in the order of a few seconds, can add enormously to the cost, complexity, and technical risk of a system. Alternatives exist that may give nearly the same benefits at much lower cost. For example, skip sequential processing of indexed sequential files permits short batch cycles—perhaps in the order of an hour or less. If this is not short enough, it is possible to provide on-line data *retrieval* without the much greater complexity that on-line *updating* requires. There are certainly cases where the extra cost of on-line updating is more than justified by substantial incremental benefits, but these tend to be quite exceptional.

The bias toward sophistication manifests itself in other ways. Designers and programmers often strive too much to achieve machine efficiency. In doing this, they ignore the broader aspects of efficiency that include programming costs, flexibility, and maintainability. We live every day with past mistakes of this sort—for example, the ludicrous situation in which a third generation computer spends its time emulating a second generation computer because earlier systems were programmed in assembly language to gain efficiency. We tend not to make this mistake now; instead, we make errors at a higher level by rejecting, for example, use of a generalized software package because of its "inefficiencies."

Failure to exploit existing technology

Paradoxically enough, while we strive for sophistication we also ignore well established technology. This is perhaps understandable in a technology that has been expanding so rapidly: it is almost impossible for the practitioner to keep up with his field. Nevertheless, it is painful to see.

A good example of the problem is our failure to include an information retrieval capability within a management information system. The ability to handle *ad hoc* inquiries has been available for well over a decade. Some companies began to apply such systems in the late 1950s, and proprietary retrieval packages came on the market soon afterwards. Although there is now widespread interest in such systems, they are used much less than they ought to be.

Poor human factors design

One of our greatest failings is the incredibly poor job we often do in serving the human user of our systems. Anyone who has tried to correct a billing error knows the problem. Systems are too unreliable, unforgiving, and uncorrectable. They spew out masses of data that are often poorly identified and displayed in a form that is difficult to comprehend. It is no wonder that users shudder a little when the computer moves in.

IS THERE A BETTER WAY?

Clearly we can do better than we have; the relatively few organizations that have done an outstanding job demonstrate what can be done.

There is no simple prescription for success in this business. Nevertheless, there are certain steps to follow that are probably necessary, but not sufficient, for success. I offer them not as a unique set of guidelines—indeed, they are all well-known—but as a reminder of those things we frequently choose to ignore.

Stay within the state of the art

It is a well-known aphorism in our trade that pioneers are distinguishable by the arrows in their backs. With rare exceptions, it does not pay to develop systems that rely on an extension of the frontier of technology. Even military systems, which have a better justification than most for such a strategy, usually are well advised to stick with proven hardware and software.

Exploit available technology

Staying within the state of the art is not a very serious limitation. The art has advanced to the point that technology does not impose many real constraints. It is perfectly feasible, for example, to design systems that have the following characteristics:

1. Considerable use of shared computer resources. Although the issue of centralized versus decentralized computation is not closed, most advanced systems will find it exceedingly attractive to combine computational loads into a few (often one) large computers. Centralization offers the undeniable advantages of economy of scale, the power of a large facility, load levelling, and the availability of a centralized data base. There will no doubt always exist systems that call for the simplicity and economy of specialization that a small decentralized computer can provide, but for the bulk of our needs we should surely look to relatively centralized facilities.
2. Considerable integration of the data base. The consolidation of files has been going on apace for a number of years. This has resulted in a reduc-

tion of redundancy and greater sharing of data across programs. There still remains, however, the need to link non-contiguous records. This only becomes feasible when one employs a fairly sophisticated data base management system for on-line storage. The capability of linking records has existed for a number of years now—beginning with General Electric's IDS—but it has yet to be widely applied. This technology has now matured to the point that it should be considered in the design of any new system.

3. Interactive computation. My earlier comment on the overemphasis given to short response time should not be interpreted as a blanket rejection of quick-response systems. Certainly the technology is available to provide reliable on-line response or file updating when the resulting benefits justify the added cost. Our strategy should not be biased in favor of either short or delayed response; rather it should aim at a *tailored* response that allows each application or user to have the response that best balances costs and benefits. Such a system will very likely provide a whole spectrum of response times—from interactive retrieval and data entry to periodic batch processing.
4. Finally, virtually any well designed system should include some decision making models. The models can vary from analytical optimizing models to man-machine decision systems that rely on elaborate simulation models. It is only when the information system provides direct support of decision making that it really begins to take advantage of the full capabilities of information technology. The most serious issue facing the designers of such systems is the extent to which the models should be embedded within the information system. It is always a difficult task to provide formal, automatic links between transaction processing and decision models. They should therefore be provided only in the case of high-volume and rapidly changing inputs and outputs; in the remaining cases the models can be loosely coupled to the transaction processing through some sort of manual intervention (e.g., smoothing or normalizing transaction data to generate inputs, and reviewing and adjusting decision outputs prior to introducing them into the operational part of the system).

Provide long-term organizational commitment

If there is anything certain about our business it is that the development of successful systems calls for long-term, sustained support. No system is ever really finished; it undergoes continual modification and adaptation over the course of many years. Eventually it may be scrapped and replaced by an entirely new system.

An organization's commitment to a long-term program should take a number of forms. One requirement is for relative stability in the resources devoted to systems development. Insofar as possible, these expenditures should not be viewed as an easy target for cash savings in time of belt tightening.

Management's commitment should also take the form of participating in the design of the system and smoothing the way for its introduction. The call for such participation has become a commonplace in our profession, but it is no less true because of that. A well designed system should become an integral part of the management process of an organization, and it is hard to see how this can come about without continual support by all levels of management. Many important management policy decisions are embedded within the MIS design, and so managers must participate to the extent required for them to resolve these issues.

A long-term commitment implies an explicit program to develop systems personnel. The need is especially critical at the management level within the systems group itself. The philosophy that any good manager can head a systems group has surely been discredited by now. The body of knowledge within the field has grown to the point that this knowledge cannot be quickly or casually acquired; the development of real professionals requires a long-term program. We should be no less reluctant to turn over systems development to the untrained and unseasoned manager than we are to trust the removal of an appendix to a barber.

An organization's commitment is also measured by the calibre of personnel they assign to the MIS function. Little success can be expected if the function is considered a dumping ground for misfits, if it does not get its share of the organization's fair haired boys, or if it does not attract outstanding computer professionals. Without at least a sprinkling of really top-flight designers, analysts, and programmers, the organization would be well advised to scale down its aspiration level to meet only bare requirements.

Emphasize users

Everyone talks about the need to serve users, but we nevertheless continue to make many of the same mistakes. Evidentally we must take much more formal and explicit steps to deal with this problem.

The use of steering committees is one way to get user participation. They can certainly help in making resource allocation decisions and in resolving policy questions. They are equally applicable at all levels in the organization—a high-level committee to deal with broad, long term issues, for example, and lower-level committees to deal with operational subsystems. The form of the committees and their tenure may vary considerably, but certainly some relatively formal means is needed as a forum for dealing with issues that cross existing organizational boundaries.

System groups should include staff members who are specifically assigned responsibility for human factors design. Included within their scope of activity should be such matters as the design of hard copy forms and CRT display formats, procedures connected with the information system (including error handling procedures), specification of the operating characteristics of terminals, and concern for the ability of the system to adapt to changing user needs. These responsibilities tend to be scattered about within most groups, rather than dealt with as a whole.

A formal user sign-off procedure should exist so that users have an explicit opportunity to get their views incorporated into the MIS design. Such sign-off should be required at all major stages of the implementation—the feasibility study, functional design, detailed design, testing procedures, and final operational acceptance.

It might be a good idea for each organization to have an ombudsman to look out for users' interests. As information systems become more and more pervasive throughout the organization, we stand some chance of being swallowed up by them unless someone is around who can pull the plug in an emergency. To be effective, the ombudsman should remain independent of the MIS group, have the authority to investigate all matters connected with the information system, and have the right to voice his opinions at the highest levels of management. One would probably not find enthusiastic support for such a position among systems directors, but it nevertheless may provide a very useful safeguard.

Devote more effort to project management

The implementation of a large MIS has itself become a formidable management task. It may require the efforts of over a hundred persons extending over the span of several years. The various subsystems tend to interact closely with one another, and therefore call for a high degree of coordination among the project teams.

An activity as complex as this requires more attention to its management than normally given. Techniques exist for dealing with the problem—network scheduling and resource allocation models, documentation and configuration management procedures, simulation packages for analyzing alternative designs, and all the rest. Unfortunately, however, most projects do not formalize project management to the extent necessary. The direct cost of good management is quite high, but the indirect cost of poor management is staggering.

Centralization of systems management

It is hazardous to generalize too much about the proper degree of centralization of systems activities. Organizations thoroughly committed to decentralized management find it hard to treat the systems activity as a centralized exception. Centralization can also increase problems of keeping the system responsive to users' needs.

Nevertheless, considerable benefits can come from a certain amount of centralization. The benefits basically stem from the economies of shared use of hardware, software, and data.

For most organizations it does not make much sense to maintain more than one group concerned with the development of standards for programming, documentation, languages, data base design, and project evaluation. Vendor hardware and software evaluation should likewise be centralized in most cases. Design of common application programs might even be centralized. Clearly we are already seeing more and more centralized computer facilities.

Centralization of this sort does not imply isolation from users. Specialized applications can—and probably should—be handled on a decentralized basis. A remote termial hooked to a centralized computer should be viewed logically by decentralized managers as an independent resource with the capabilities of a large machine at a fraction of its full cost. To be sure, any centralization imposes some constraints on the organization's subunits, but when intelligently applied the constraints still can leave ample room to meet users' requirements. After all, we all must live within certain constraints, whether imposed internally or externally by governments or vendors.

Incremental development

A comprehensive system takes many years to develop. No one can foresee how technology and users' requirements will evolve over this length of time. It is therefore absolutely essential that systems be designed in a way that permits continual modification and extension.

A number of steps can be taken to achieve a much higher degree of flexibility than most systems offer. Perhaps the most important requirement is a master plan to guide the development of separate subsystems. Such a plan defines the subsystems that will comprise the eventual overall system, and establishes the major interfaces among these subsystems. It is only when such a plan has been developed that one can have any assurance that all of the subsystems will mesh together.

The master plan need not go into great detail. Its purpose is to establish constraints within which each subsystem can be developed more or less independently. In principle, one should limit the detail to just that level required to define the interfaces among subsystems. Detailed work beyond this level can wait until the actual implementation of the separate subsystems.

Insofar as possible, the subsystems should be implemented in a sequence that provides early interim benefits. Certain technological constraints may partially limit the choice of sequence—for example, some generalized software may be required before application programs can be written, and a forecasting program cannot be implemented until the order entry subsystem from which it gets its inputs has first been developed. Other factors, such as the particular skills available and political considerations, may play a part in choosing the implementation sequence. The basic strategy should be, however, to implement each subsystem in a way that provides interim benefits that justify development costs, while at the same time leading toward the eventual accomplishment of the master plan.

The plan should evolve along with the system: it is not meant to impose inflexible constraints on the design. Changes will inevitably occur as technology advances, the organization's needs change, and the system matures. The changes should, of course, be controlled and consideration given to trade-offs and the need for coordination among subsystems.

In addition to the development of an evolutionary master plan, other well-known techniques exist for increasing the flexibility of a subsystem. These include use of higher-level languages, modularity, data and program independence, avoidance of unnecessary program complexity, and careful documentation. These techniques typically carry a short-term price, but they are vital for long-term success.

CAN SUCCESS BE REPLICATED?

We know that real success can be achieved, but we also know that it is relatively rare. This is so because success requires the rare combination of exceptional management, sustained support, and top-flight technical personnel. Without these ingredients, results are likely to be marginal at best and disastrous at worst.

Unfortunately, there is not enough talent to insure that each organization has an adequate supply. The great challenge we face is to develop a technology that does not require genius to implement. Conversion of a field from an art to a routine science is the chief sign of its maturity. To achieve this in our field we must devote a great deal more effort to gaining a better theoretical understanding of complex systems, planning and control of large organizations, and the behavioral aspects of formal information systems.

MIS technology—A view of the future

by CHARLES H. KRIEBEL*

Carnegie-Mellon Universiy
Pittsburgh, Pennsylvaina

INTRODUCTION

Industry tabulations for 1970 indicated there were some 85,000 computer installations in the world, valued in excess of $40 billion. By 1975 these figures are expected to double; in ten years they may be redoubled. The continuing exponential growth in raw computing power prompted Art Buchwald recently to editoralize on "The Great Data Famine" of the 1970's. According to Buchwald's expert source by January 12, 1976 ". . . every last bit of data in the world will have been fed into a machine and an information famine will follow." To cope with this impending disaster, a crash program is urged in which (1) "no computer can be plugged in more than three hours a day"; (2) the government will spend $50 billion to set up data manufacturing plants and their output will be mixed with soy bean production; and finally (3) a birth control program will be advocated for all computers—provided, "the Vatican's computer gives us its blessing."

On Tuesday evening, September 9, 1965, the largest power failure in history blacked out all of New York City and some 80,000 square miles of neighboring areas in which nearly 25 million people live and work. The blackout lasted for three hours before partial power was restored. It is awesome today to imagine what the consequences would be if a computer blackout were to occur throughout the world. Yet in spite of our increasing dependency over the past twenty years on computer technology, one continues to hear of a "technology gap." Instead of a "data famine" the technological issue today is that computers haven't been able to do all that was expected of them. In stronger language, Isaac Auerbach stated at the closing session of the recent fourth triennial Congress of the International Federation for Information Processing that the performance obtained by users of computers for business data processing shows: 20 percent are successful, 40 percent are marginal, and 40 percent are failures. Reviews by others of computers in management information systems also sharply criticize the notable lack of results in spite of great expectations.[2] If true, this is a dismal return for a $40 billion investment in equipment. What went wrong? Who's at fault?

On the one hand computer manufacturers and the industry have been blamed for overselling, for not developing the right technology, and for not solving the users' problems. On the other hand the user and managers have been criticized for overbuying hardware, for misapplying an inert technology, for "living in the last century" and not exploiting the modern methods of today. There is some truth on both sides of these arguments. But acknowledging the existence of a gap is at best only the beginning of a solution. What are the dimensions of "the gap?" What remedies are proposed to close it? What future developments can we anticipate towards progress in MIS? Before gazing into a crystal ball for the answers, we can gain perspective on the problem by briefly probing the surface of the issues involved. A conceptual framework is particularly important in this case because the broad connotation of MIS and misunderstanding are the source of most of today's difficulties.

INFORMATION STRUCTURES AND MIS

The phrase "management information systems" or "MIS" has gained popularity as a descriptor for information processing activities in support of management. If one explores the environment of management it is soon apparent that decision making is the primary function that distinguishes managerial activity from other behaviors. In fact, management decision occupies a singular role in management literature, since other

* The author wishes to acknowledge many enlightening conversations on MIS with Dr. David B. Hertz and R. George Glaser, Jr. of McKinsey and Company, Inc.

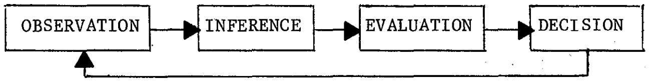


Figure 1—Basic stages of activity in decision processes

behavior—such as planning and control—is often defined in terms of decision activity. Thus, an understanding of the central issues in MIS might logically begin with the concept of management decision.

To economists decision making is the allocation of scarce resources. Other writers emphasize the choice activity in that for them, decision making is selecting the "best" action from among available alternatives—where the criterion of choice might also be viewed as one of the alternatives. Since our immediate interest concerns informational aspects of decision, it is instructive to adopt a broader view of the process by considering the stages which rationally precede the choice of action and a commitment of resources.

Although a variety of elegant models are available in the literature, the essential ideas behind a rational structure for decision making are really quite simple. A parsimonious description of the decision process would include four identifiable stages of activity: observation, inference, evaluation and choice (see Figure 1).* Within the limits of this model, the process begins with observing states of the environment; surrogate images are assigned to natural events, measurements are taken, and data is collected, encoded and stored. The second stage in the process involves analysis of the recorded data as a basis for inference and prediction. Here, in effect, reports are made and reviewed to determine whether or not a situation requires a decision to be made. Stage three is initiated by the need for a decision. At this point inferences and projections are analyzed to identify action alternatives and these in turn are evaluated with respect to goals, constraints and efficiency. Finally, an alternative is selected which is preferred (or "best") under the criterion of choice, the action is communicated, resources are committed, and the decision is implemented. As implied by the diagram the process actually perpetuates itself through cycling and feedback, since upon implementation the decision becomes a source for new observation; and so on.

Now suppose we consider the development of information structures in support of the activities involved in each stage of this process.* That is, the simplistic decision model can serve to characterize four levels of the relative maturity (or sophistication) achieved by an information system. In its most elemental form an information system is simply a repository or data bank, encompassing only the "observation" activities of the model. The raw data has been organized in some convenient manner and is stored, say, in a computer file; upon interrogation the manager retrieves a report and performs all subsequent analyses from inference to decision. Most accounting systems are of this elementary form, employing a linear transactions model of the organization and serving as simple data banks. Simplicity, *per se*, is not a criticism of such systems; their shortcoming in application is that typically they do not discriminate between vital information and trivia. Thus managers interacting with an elementary data bank system often become frustrated by the lack of relevancy in large volumes of data.

At the second level of maturity in development the data bank system has been expanded to include most of the activities of inference as part of the formal system. In addition to exception reporting, the system now includes the capacity to forecast and to artificially test the consequences of implicit decisions. The emergence of the fields of management science and operations research accompanied by computer-based models, particularly linear programming and simulation, has done much to facilitate the realization of these selective-inference information systems. Here the manager can interrogate the system with "What if . . ." questions, and receive an array of consequences in response for his evaluation. The hidden danger in this dialogue is that the manager is usually insulated from factual data in the system by a host of assumptions imbedded in the model which provides the inferences. While a seemingly obvious caution, it is surprising how readily individuals are lulled into believing "realistic-looking" output is fact.

At level three in development, evaluation activities have been programmed into the selective-inference structures so that the information system now encompasses action recommendation. Here the need for a decision is triggered within the formal system on the basis of monitored observations and predetermined rules or a time scheduled event. Procedures are programmed to evaluate alternatives against assigned goals as the situation requires, the "best" course of action is chosen, and the recommendation is communicated to the manager. At that point the manager

* The well-known antecedent for this model is, of course, the "scientific method." For an interesting and more elaborate taxonomy see Simon (1960).

* Analogies of this form have been proposed by others in similar contexts; for example, Chapter 15 in Gregory and Van Horn (1963) and Mason (1970), in particular.

either implements a decision based on the recommendation or he rejects the alternative and further analysis may be initiated. The most common form of this action-recommending information system in organizations today is the advisory staff group or committee. In this case line management delegates authority for a certain area of responsibility to a staff department, but retains final control for decision through review and certification. Another variant of this system has appeared in the form of large scale optimization models, particularly linear programming applications in industrial scheduling. In many of these cases, however, systems originally designed for "action recommendation" have over time reverted back to "selective-inference" systems as limitations of the model became apparent.

In the final stage of maturity the entire decision process has been automated within the information system. All activities from observation to choice and the ability to initiate action, commit resources and monitor results have been programmed. In effect the manager is now outside the structure having fully delegated his authority, although he retains the power to revoke the delegation at some future time. The simplest form of the automated decision system is the "standard operating procedure" in organizations; a more sophisticated example would be a computerized process control system in a petroleum refinery. Modern factory inventory control systems are another common example. Obviously, automated decision systems require extensive knowledge of the decision process for the given application and consequently their development has been limited to well-defined environments.

Figure 2 summarizes this overview of information structures in MIS. Concerning the classification it is important to recognize that the levels of development should not suggest a linear or necessarily sequential progression of system maturity. That is, the MIS in a modern organization today is not one of these information structures; it is a composite of all four types. Said differently, the two basic components of design in any information system are (1) a *model* of the user, and (2) a *data* store. The former defines the output interface or system application, and the latter constitutes the environmental image or system input. For MIS the user model reference is management decision processes. In the evolution from "data bank" to "automated decision" the user model becomes technologically more sophisticated, as more activities of the process are formalized and programmed into the system. But decision processes exist at all management levels in an organization hierarchy. Thus, one finds information structures of different maturity at different management levels. The most mature computer-automated information structures in MIS today are still confined to lower management levels.

Level of Development	*Information System Type*	*Decision Activity Model (Fig. 1) Formalized and Programmed*
1	Databank	Observation
2	Selective-Inference	Observation + Inference
3	Action Recommendation	Observation + Inference + Evaluation
4	Automated Decision	(All above) + Decision + Feedback

Figure 2—Classification of information structures in MIS

The preceding discussion provides a framework in which to assess the dimensions of MIS technology, the "gaps" today inhibiting implementation, and the direction of future developments. For present purposes I will confine these remarks to two dimensions: (1) the user model component and (2) the data (processing) store component.*

USER MODEL TECHNOLOGY IN MIS

The lackluster results vis-a-vis expectations realized in MIS to date can be attributed in part to the fact that the implicit model of the user has remained naive. As with more elementary information storage and retrieval systems, the user of a MIS can occupy two distinct roles either as a generator of system input or as a consumer of system output. In most cases these roles occur in different individuals in the organization and each interacts with the system differently. The structural interface of the system which facilitates communication between these respective roles is the data file.

Dating from the early 1950's the user model which typified first generation, computer-based MIS was *accounting* transactions. For most applications this model better served the system input role than the output requirements of managers. Consequently, management decision support was limited to a class of data bank structures for financial reporting, and system performance evaluation (when executed) was based on administrative (or clerical) cost reduction. With the development of operations research techniques and the evolution of second generation computer systems for MIS, accounting was supplemented by a series of user models based on *management* function, such as personnel administration, production-inventory control, mar-

* In Chapter 1 of Kriebel, et al. (1971) we have classified MIS research along five dimensions, however, space limitations here prevent a more extensive discussion.

keting management, and so on. In seeking to close the output gap in the user model and advance to selective-inference information structures, functional area managers in effect developed their own information systems. The counter-cries to this movement were the appeals for "integrated data processing" or (in some circles) "the total systems approach." System compartmentalization during this period aggravated a variety of development problems including data collection, data redundancy and inconsistency, and serious limitations on file cross-talk. System costs were often hidden in overhead accounts, data processing feasibility governed applications selection, and "management intangibles" dominated systems evaluation.

As total costs continued to increase in spite of gains in computer hardware efficiencies, the third generation systems of the mid-to-late 1960's saw a shift in emphasis towards consolidation—though not necessarily of the "total systems" variety. The output-oriented, management function user models in large measure were abandoned for an input-oriented *data base* model with the goal of data processing efficiency. The development orientation in MIS now focused on a corporate data bank which stressed input data format, flows, and files with little attention to the specific output requirements of end users. Working applications programs in the system were decoupled from data files through the imposition of file management software. Most formal information systems today are still in this stage of evolution.

From this brief history one sees that the user model technology in MIS has not progressed very far over the past fifteen years. At lower management operating decision levels where an OR model may be in good correspondence to the actual management problem, the decision mechanism often becomes an automated information structure in the MIS. But above the level of routine behavior, the user (output) model rapidly returns to a data bank structure. Today's "solution" to the issue of user output requirements is to maximize the flexibility for options at the output interface. Data managers and report generators are two common examples of such output flexibility. In effect specification of the user (output) model is now left up to "the user." Similarly, evaluation of the system benefit to the user is becoming *his* own responsibility. That is, in several large organizations today the computer-based resources of the MIS have been divisionalized into a "profit center" and, though not a freely competitive market place, the user is required to pay for services requested. The performance criterion for system operations is thus "simplified" to efficient utilization of resources.

This present stage of evolution raises several questions concerning future developments in the user model dimension of MIS technology. For example, the profit center organization is a move toward the information utility concept; but is "information processing" a homogeneous commodity? Divorcing the system design from specific user needs is an attempt to make the technology independent; but is MIS technology inert? The contemporary orientation on data input and EDP technology implies that resource efficiency is a major bottleneck; but has EDP utilization been the key barrier in MIS implementation? The data base model development of a MIS seeks to establish a corporate data bank, often building from the bottom up; but do information structures for managers draw upon common data?

In anticipating the next ten years, my first prediction is the demise of the present-day data base orientation as the dominant user model image in MIS development. Instead the development focus will shift back to an output orientation that is user dependent, though not as a return to the second generation MIS. The emphasis will be on basic decision processes and problems of the specific organization and manager without regard to the functional environment. Information structures will be developed to solve a particular manager's problems; some of the structures will be generalizable to other situations in broad outline. For example, one early attempt at such a structure for strategic planning decisions has been the so-called corporate economic simulation model.[14] The mixed results achieved by corporate simulators thus far can be attributed in part to the difficulty in implementation of securing top management attention and involvement. Another less publicized illustration of an information structure tailored to a specific decision problem is the concept of an "issue map" developed by McKinsey and Company for Mayor Lindsay of New York City.[7] Still another structure that I would include in this context was the original development and application of PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method).

The user model image that focuses on *critical decision* processes for MIS development, however, is only one aspect of the new technology. Allied with the change in orientation will be a rise in the application of operations research and behavioral science in developing and implementing these information structures. It is common knowledge that many of the basic methods and known results in operations research are not being used by managers today, even though many of the techniques have considerable potential as decision aids.[8] This issue has been labeled a "communications gap," although some feel the problem is more fundamental.[3] I think a majority would agree that MIS technology is not inert, even though the computer might be. To achieve compatibility at the user/system (or man/machine) interface, technology can either (1) adapt the system to fit the user's requirements (or behavior patterns), or (2) seek to change user behavior to better match system capabilities.* Much of the effort by industry professionals to date has been directed at the second approach, particularly in the popularized education programs on "OR and computer concepts for executives." Results notwithstanding, the empirical evidence on MIS implementation suggests: (1) except for the most routine activities, managers actively resist attempts (real or perceived) to erode their authority base; (2) the executive's orientation is toward the "people" in the organization, his self-concept is as an individual who directs the activities of others; (3) there is relatively little understanding of upper management decision processes. Emerging information structures in the next decade will increasingly incorporate behavioral parameters in their design which reflect organizational associations, leadership style, and management attitudes as a means toward improving acceptance and effectiveness at the user/system interface.

For example, in the mid 1960's the experience of IBM with "executive terminals" for their own MIS led to the establishment of an Information Center at corporate headquarters with a human rather than a machine interface for the user, primarily because of dominating behavioral considerations (cf., Chapter 8 in Reference 10). Similarly, at Alcoa in developing their new Picturephone Remote Information System (APRIS) for executives, "the principal design criterion was ease of use . . . rather than provide just a tool, the goal was to provide . . . the service of better access to information."[4] The "complete" user guide for APRIS is printed on a 3″×5″ index card. More basically, perhaps, research is already under way to parameterize behavioral constructs in the decision process, such as "participation,"[16] to model how managers identify problems to be solved,[13] and to understand how humans process information in the act of solving problems.[12] Before the end of the decade I think these developments will impact on MIS design in the appearance of more "personalized" information structures which better match the users' needs and in the identification of new requirements for system technology. Within the data processing industry the emerging trend toward companies which provide "information services" will continue; they will be the developers of the new technology which solves the user's problems.

DATA PROCESSING TECHNOLOGY IN MIS

Although organizational information systems existed long before the appearance of the computer and EDP, the formal designation MIS is commonly associated with the modern information technology of the last twenty years. A company's decision to "establish" an MIS capability in this period contained two conditions of entry: (1) a large initial capital investment of $.5 million or more, and (2) a non-revocable commitment to EDP, since the "bridges" to existing manual systems were usually "burned" as new systems were developed and mechanized. Consequently, the initial stakes to play the MIS game were high even if amateur players did not always recognize them. But then so was the promise of payoff. The economic equation for management was simple: replace labor with capital so that as data volumes increase and technology improves the per unit cost of processing declines. The engine that would power this system (or the MIS) was the computer.

Computers have been successfully employed in engineering instrumentation, scientific data processing, military and space applications, and in business where the application involves large volumes of data and relatively straightforward computations, such as in communications switching or bank check processing. To secure the capital leverage of the computer in each case requires that the process involved be *formalized* and programmed. But the needs for different applications are different and, as discussed above, the absolute requirement to formalize management decision processes has been the key barrier to the computer's success in MIS. In my opinion this problem has been further compounded by the promoters of so-called "integrated systems."

EDP technology encompasses a broad domain beyond "the computer," from firmware to peripheral devices and communications, from POLs to data management software and hypervisors, from system configuration to systems analysis and people. Within the space available it is obviously impossible to cover all aspects or even do partial justice to the many contemporary developments. As a compromise, I will instead highlight a subset of the whole that I think holds particular significance for MIS in the future.

* I'm reminded here of the lament some years ago, by the EDP manager of a very large steel company who in responding to an inquiry by the Chairman of the Board was required to process some 90 million records over a two-day computer run. Apparently, the Chairman did not ask his "MIS" the *right* question.

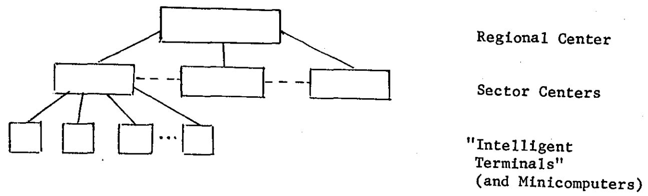


Figure 3—Distributed data processing grid

Perhaps the most distinguishing characteristic of third-generation EDP system operations was the emergence of teleprocessing, facilitated by multiprogramming and data communications technology. Although early time-sharing applications were restricted by the existing hardware and operating systems, considerable progress has been made in a relatively short time span. This development is significant for at least two reasons. First, it provided the basic framework for distributed data processing; second, it turned the centralized vs. decentralized operations debate into an academic issue to be decided on a relative basis by organizational philosophy. The distributed data processing (DDP) approach (see Figure 3) employs a hierarchy of EDP centers, ordered on the dimension of capacity and interconnected through direct communication lines. (Increasingly in the future these lines will be dedicated for data transmission rather than modifications of voice networks.) At the lowest level in the DDP grid the point-of-origin device for data capture and interaction will be an "intelligent terminal," ranging from a Picturephone-like* device to a minicomputer with local data processing capacity. Where online access and speed are not important data cassettes will provide communication linkages. In more sophisticated applications computer networking will be employed for direct communication between computers and multiprocessor centers. Although thus far the DDP concept has been employed primarily in experimental systems, I believe it will be commonplace in the future.

The distributed data processing approach in one sense is a natural outgrowth of the time-sharing principle to extend system utilization while capturing some of the economies-of-scale in hardware technology. However, experience has shown that the massive "universal" system approach rapidly leads to substantial diseconomies in software overhead and administration. By analogy, the computer is not a universal machine; "a truck, a motorcycle or a racing car each has a different engine which is designed for high performance and economy in the specific use intended."[1] The computer system and MIS that seeks to be all things to all people in one package is doomed to failure at the outset. The cost balance in configuration is between operating control over a fragmentation of diverse specialized systems which may be locally optimal but globally expensive, and the universal system "dinosaur."[9]

A critical problem to date has been technically with the integrated data processing approach—particularly in extreme applications for MIS which have attempted to "integrate" everything from customer sales transactions to the President's appointment calendar. The basic difference under DDP is not to integrate but instead to provide a well-defined interface for relatively independent sub-systems. Data managers have been a first step and in the future these systems will be dedicated to sub-set applications as required. What emerges then is a *portfolio* of technology (packages) optimized with respect to the portfolio of user information structures. Some of the elements in the overall portfolio will interact, some will not; however, all elements will be highly modular. That is, one application may periodically require augmented computer capacity from the sector or regional centers in "competition" with other applications, others may share certain master files for data, and still others may be locally self-sufficient. For example, the "executive terminal" might well become an information structure serviced by a minicomputer with limited communication to a corporate data bank via cassette.

The utility structure of the DDP grid recalls the previous question: Is "information processing" a homogeneous commodity? The purist would answer affirmatively at some level of computer science or, above the ISP (Instruction Set Processor) level, one could identify kernel processes. But the absolute answer to this question does not seem critical to me. That is, defining the requisite interfaces will begin to establish the standardization required to achieve plugboard modularity of sub-systems. For example, General Electric is currently addressing parts of this problem with "interprocessing" in which customers' computers and GE's network service are used jointly[5]—to cite one illustration. A more interesting issue, perhaps, for managers and EDP technology in future MIS now becomes the "make or buy" decision.

Earlier I remarked that the economic rationale by management in making the commitment to EDP and MIS was essentially to replace labor with capital and improve productivity. What has happened, however, is that total costs have increased. Even if one examines only the *apparent* costs of systems operations and development the capital/labor ratio advantage has

* Picturephone is a registered trademark of the Bell System.

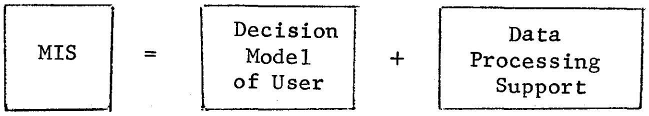
not materialized; equipment hardware cost continues to decline as a percentage of the total which includes systems personnel and administration—today ranging between 35 and 45 percent. Furthermore, as most of the trade literature reports, apparent costs are only the visible portion of the economic "iceberg" in MIS when we include such *hidden* cost factors as the drain on management time, inadequate priority analysis of competing applications (and the foregone opportunities), abandoned projects due to vacillating support, security in the EDP department, inefficient software, etc. The constant advance in the sophistication of the new technology will continue to raise the economic stakes for payoff in MIS as this total cost base expands. Sunk cost notwithstanding, the decision many companies are literally beginning to face is whether or not they can afford to be in the MIS business.

This economic crossroad in MIS is not to suggest that management is contemplating a nostalgic return to the Dark Ages. More realistically, they are weighing the practicality of making it themselves or buying it on the outside. Today information services companies sell pieces of an MIS; you can buy information problem solutions as a complete package and for a price contracted *in advance*. Often this opportunity does not exist inhouse and cost estimates are notoriously biased. The large corporations with the requisite capital base and absolute system constraints will retain inhouse technology for their MIS. Increasing numbers of companies in the $50 million annual sales or less category, however, are going to buy their MIS technology over the next ten years. Few companies today generate their own electrical power requirements; many companies buy their legal services as needed; most small companies buy their accounting services. The distributed data processing approach will facilitate a comparable opportunity for EDP technology. Management will decide how much of it to "buy" and how much to "make" in establishing an economically viable MIS. For the data processing industry: establishing the centers, developing the technology, and regulating operations will be determined in the market place by the user or the government or both.

CONCLUSIONS

Having reviewed the state-of-the-art in MIS technology contemporary developments indicate that the future holds no radical departure in the promise of a major technological breakthrough, although there will be change. The MIS as the computer itself is still in its infancy and progress in design and development is an evolutionary process that depends on how much is learned from past mistakes.

Perhaps the most significant change in the future from the perspective of results will be a reestablishment of purpose as information structures in MIS seek to solve the bona fide problems of users. In simplified terms we characterized system technology by the model:



We further stressed that the many information structures which in aggregate constitute a MIS range in their sophistication as a descriptor of management decision processes from a relatively naive "data bank" model to a complete and closed-loop program for "automated decision." As developers seek to advance the state of user model technology the MIS design focus will shift from a data base orientation to a focus on the critical decisions of line managers in organizations. More emphasis will be placed on education of both system analysts and managers to close the communications gap; design teams will be formed to better utilize the methods of operations research and management science. To further increase understanding and acceptance behavioral considerations will assume a dominant position in development, becoming parameterized where possible to better match the information structure to the user's problem. The MIS then will become a portfolio of information structures designed to solve the information requirements of a hierarchy of management decision processes.

Data processing support within MIS will also undergo change over the next decade in an evolution from current developments. One major outgrowth will be extension of the distributed data processing approach to delivering a portfolio of EDP support in MIS. Within this framework efforts on integration will diminish in favor of modularity in design to decouple and develop relatively independent sub-systems with well-defined interfaces for plugboard compatibility. Data management systems will become dedicated to improve the linkage between data base development and the user model portfolio's requirements. Communication within the distributed data processing grid will be over dedicated lines for data transmission or in some cases by data cassettes. The minicomputer will become commonplace as "intelligent terminals" in the grid. Finally, pressured by rising inhouse costs and the requirement for economic payoff, all but the large corporations will decide to buy their EDP technology from outside suppliers, in some cases retaining minicomputer terminals as the system connection.

In looking toward the future of MIS technology then, my view sees *payoff* and *portfolio* as the most descriptive adjectives. To me they are inevitable if MIS in the future is to include *progress*.

REFERENCES

1 I L AUERBACH
Talk presented at closing session IFIP Congress 71
Ljubljana Yugoslavia August 23-28 1971

2 R W BEMER ED
Computers and crisis
Association for Computing Machinery 1971

3 F W CHURCHMAN A H SCHAINBLATT
The researcher and the manager: A dialectic of implementation
Management Science February 1965

4 M L COLEMAN K W HINKELMAN
W J KOLECHTA
Alcoa picturephone remote information system: APRIS
Presentation at the Fall Joint Computer Conference Las Vegas Nevada November 16-18 1971

5 G FEENEY
A three-stage theory of evolution for the sharing of computer power
Computer Decision November 1971

6 R H GREGORY R L VAN HORN
Automatic data processing
Second Edition Wadsworth 1963

7 D B HERTZ
Systems analysis in urban affairs
Speech delivered in New York City October 19 1967

8 D B HERTZ
Has management science reached a dead end?
Innovation 25 1971

9 P HUGGINS
Comdr Hopper tells users "Dinosaur systems have to go"
Computerworld June 16 1971

10 C H KRIEBEL R L VAN HORN
J T HEAMES ED
Management information systems—Progress and perspectives
Carnegie Press 1971

11 R MASON
Management information systems—What they are, what they ought to be
Innovation 1970

12 A NEWELL H A SIMON
Human problem solving
Prentice Hall 1972

13 W F POUNDS
The process of problem finding
Industrial Management Review Fall 1969

14 A N SCHREIBER ed
Corporate simulation models
Graduate School of Business Administration University of Washington Seattle Washington 1970

15 H A SIMON
The new science of management decision
Harper Brothers 1960

16 V H VROOM P W YETTON
Models of participation in decision-making
Novus December 1971

Selective security capabilities in ASAP— A file management system

by RICHARD CONWAY, WILLIAM MAXWELL
and HOWARD MORGAN

Cornell University
Ithaca, New York

Although the general requirements of data security in file management and retrieval systems are fairly obvious, and have been often stated[1,2] current practice does not provide even an approximation of these requirements and, in fact, current languages and operating systems, with few exceptions[3,4] make it relatively difficult to achieve these objectives when one tries to do so. Apparently, current languages and hardware serve as an informal security and privacy protection system that is more or less satisfactory for most current applications. While security and privacy controls are often discussed, they do not appear to be high on most managers' lists of pressing problems.

The authors' concern with security is limited to the systems implications which arise when one or both of the following circumstances exist:

1. A common file is used for two or more different purposes, e.g., an integrated management information system.
2. A file is directly accessible by those who are responsible for generating input, and those who use the information it contains.

Questions of physical security of facilities, tampering with hardware etc., have been treated elsewhere[5,6] and will not be discussed here.

At present, the programming languages and operating systems that are used for most file maintenance and information retrieval applications are sufficiently complex and esoteric that professional programmers and/or production controllers have been interposed between the ultimate users of the file and the file itself. Moreover, the majority of such applications are still handled by relatively conventional batch processing systems, without remote access devices being placed directly in the hands of the information supplier or consumer. Both the physical separation, and the necessity of human review of access to the file serve to provide an informal but adequate security system to protect against abuse by the information consumer. The programmers and production controllers in such a system, however, often have considerable freedom to attack the security or privacy from within. (How many data processing departments are able to keep salary information from their programmers?)

It is clear that remote access systems represent the future for such applications—the only question being how rapidly the future will arrive. It is also clear that user and problem-oriented languages now exist[7] and once data-consumers have experienced the power and convenience of these languages they will not willingly return to a situation in which they are at the mercy of a staff programmer. Both of these developments suggest that formal and explicit treatment of the questions of data security is rapidly becoming a necessity. The following describes the approach to these questions that has been taken by the authors in ASAP, a file maintenance and information retrieval system designed and implemented by the authors.

There are three principal questions associated with the security of a particular file:

1. Which people with physical access to the hardware containing the file should have access to any particular file?
2. What subset of the file is each authorized user permitted to access?
3. What types of processing operations are permitted to each authorized user?

ASAP approaches each of these questions by using a complete directory of authorized users which is appended to the master file in question. While ASAP cannot prevent the data-set that represents the master file from being processed by independent programs written in other languages, it can and does supervise all requests for information entry, update, or retrieval which are written in the ASAP language, and in doing so enforces the security restrictions that are explicit in the user directory. This directory contains, for each authorized user:

1. "password" identification information.
2. A description of the file subset accessible to this user.
3. A description of the processing actions this user is permitted.

RECORD SELECTION

The first method of specifying the subset accessible to a particular user is to describe which records of the file are accessible to him. This is done by including in the directory entry for each user a Boolean expression that describes by content those records to which he is permitted access. When a user makes a request, a monitoring routine based upon this filed Boolean expression is interposed between the file management utility routine and the user program. The result is unobtrusive, but absolute—the user program only "sees" records for which the Boolean expression is satisfied; other records essentially do not exist for this user. The syntax of ASAP permits the user to easily further restrict the selection of records from the file, but this individual Boolean constraint is automatically and invisibly 'ANDed' to the selection criterion for each and every request submitted by this user and there is no way he can override it without going completely outside of ASAP and obtaining independent access to the master file.

For example, consider a master student file at a university in which there is one record for each student. One user of this file might be the administrative office of the College of Engineering. (The system is, of course, quite content with such 'corporate' users, actually representing a number of different individuals.) The selection condition for this user would probably be something like:

```
COLLEGE='ENG'
```

where COLLEGE is the name of a field in the record and 'ENG' is one of the possible values of the field. The user representing the office of the Dean of Women might be assigned the selection restriction:

```
SEX='F'
```

and a user concerned with those football players on athletic scholarships who are in academic difficulties might be restricted with:

```
ATHLETICS='F'
              AND FINANCIAL_AID>1000.00
              AND GRADE_AVG<2.0
```

A complete program in ASAP could consist of the following statement:

```
FOR ALL STUDENTS WITH CLASS='FR',
  PRINT LIST: NAME, CAMPUS_ADDRESS,
              CLASS_RANK, ORDERED
              BY CLASS_RANK;
```

If this request were submitted by the College of Engineering it would produce a listing of all freshmen engineering students. Exactly the same request submitted by the Dean of Women would yield a list of all freshmen women.

While the primary purpose of providing this mechanism is to restrict the distribution of information to those users with a preestablished need-to-know, it has a secondary effect which may be more important and interesting than the security aspect for many applications. This is the fact that each user can deal with his accessible subset of the file exactly as if that constituted the complete file. He is in every sense unaware of the existence of other records and can employ the full facility of the language and the system for his subfile, without requiring any special treatment because it is a subfile. This means that most of the difficulties associated with multiple, decentralized files can be avoided without sacrificing their simplicity of use. One can avoid the currently typical situation in which an individual student is the possessor of a score or more records in different and separated files in various stages of update and decay.*

FIELD SECURITY

A second method of defining the subset of the file accessible to a particular user is to assign different 'security classes' to different fields within each record. When the record is defined each data element (field) is assigned to one of nine security classes. One class is called unrestricted and this is the default when no

* In conventional processing of sequential files the 'invisible' records in such a system would impose a prohibitive processing penalty on the user of a small subset. This is avoided in ASAP by simultaneously processing many independent requests in one pass through the master file.

explicit security designation is given; the others are designated 1, 2 . . . 8. The classes are independent, with no hierarchy implied by the level numbers.

When each user is enrolled in the system the security classes to which he has access are explicitly listed. While every user has access to unrestricted information, access to each of the other three classes must be explicitly granted or denied. Field and record security are multiplicative—a user may see only authorized fields in authorized records.

To continue the previous example, the fields of the student record might be classified in the following way:

Unrestricted information:
Name, campus and home address, telephone, college, class.
Security Class 1—Personal/biographical information:
Medical history, draft status, disciplinary actions, race, religious preference.
Security Class 2—Academic information:
Class standing, individual course records, standardized test scores.
Security Class 3—Financial information:
Treasurer's account, financial aid, family financial report.

Now the administrative office of the College of Engineering, whose access is already limited to records for which COLLEGE = 'ENG', could also be limited to the fields in those records that are unrestricted and class two information. Fields containing personal and financial information, as well as the complete records for students not in engineering are inaccessible to this user.

As in the case of record, a significant advantage of the system is the opportunity to combine several different types of information in a single master file, and keep that information out of reach of unauthorized users, out of the way of users who don't want it, and still have complete information readily available when it is required.

RESTRICTION OF PROCESSING OPERATIONS

A third, almost independent type of security concerns the type of actions that a particular user can perform. Actions are classified in the following way in the ASAP system:

1. Read only access
2. Read/write access to existing records
3. Record creation and deletion
4. Ability to extend language by adding definitions*
5. Attach read only external procedures
6. Attach read/write external procedures
7. Alter the record definition
8. Modify the user directory—add or delete users, or change the security restrictions for existing users

The directory entry for each user includes a list of the types of actions that are permitted to this particular user. Read only access to the accessible fields of accessible records is assumed; all other types of actions must be explicitly authorized. No hierarchy is implied in the above list so that, for example, the ability to extend the language does not imply write access to the file.† Obviously, the ability to modify the user list must be granted with great frugality since this implies the ability *to obviate the entire security* system.

IMPLEMENTATION

ASAP is a compile-and-go system—each run begins from source, and security tests are applied at the source language level. Since no object decks are ever produced by the system, there can be no opportunity for security to be obviated by an alteration to the object program. Although the compile-and-go strategy is unusual in the data processing environment, it does not exact the penalty in compilation time that might be expected. The compiler is very fast, and recompiles source in time comparable to, if not better than, time required to load the object modules of a corresponding program. (On a 360/40 ASAP compiles at card reader speed, on a 360/65 at 50-100 statements per second.)

The first card of each user program is an identification card bearing the user's password. This is used to search the user directory for the appropriate entry. When the user's entry is found, the record selection condition is passed, in source language form, to the compiler, along with two bit masks, one for field security, and one which specifies permissible actions.

The record selection condition is compiled into object code that is appended to the file management utility routines. This code is exercised for each record of the master file, thus representing some execution overhead. Since the compiled code is made part of each

*,† In ASAP, users may define report formats, input formats, codes, and other frequently used constructs. These are then catalogued for future reference. Often, therefore, users will extend the language by adding special reports or macros to the system, an action quite independent of the ability to write on the master file.

selection criterion, however, this often amounts to no more than two or three machine instructions per record. Only records that satisfy these conditions are passed on to the program representing the user-specified record selection criterion.

Both field selection and action restriction are entirely exercised at compilation time and represent no execution overhead whatever. Each entry in the symbol table, e.g., field name, report name, keywords, etc., contains a security mask. On each source reference to a field, the security class of that field is ANDed with a copy of the current user's security mask, and a simple test then decides whether or not the field is inaccessible to the user. If an attempt is made to reference a "classified" field, a condition code is set that the run is terminated after compilation. When processing definitions of entities which may include information from several fields, such as reports or summaries, the security of the report is simply the OR of the security classes of all fields mentioned. Thus, while each field of a record may only have a single security class, a report may require the user to have access to several classes before he can use it.

The permissible action checking is done in a similar manner. Each action is tested against the action mask for the user for whom the code is being generated, and an improper action will cause the run to terminate after compilation.

Since references to the symbol table are made for every symbol scanned, the overhead involved in checking the security classification is rather small. And, since it is only exercised during compilation, at rather low cost, a fairly large set of security restrictions can be enforced.

The different types of security are not quite independent in the ASAP system, although there is no technical reason why they could not be. When a user is permitted to attach an external procedure (which may be written in any programming language) the system does not attempt to enforce field security. Record security is still effective since the call to the external procedure is not made until after the record has been selected for processing. However, once the record is selected, the entire record is available to the external procedure. Although it would have been possible to pass an abbreviated copy of the record in which inaccessible fields had been blanked out, the substantial overhead this would require was considered unnecessary, since the use of external procedures requires a special security class which must be approved by the system controller.

The current implementation also includes a provision for encrypting the master files. This is performed using standard cryptographic techniques, with the keywords based on the system controller's passwords. The cost for encrypting or decrypting a 500 byte record on a 370/155 is about 500 microseconds of CPU time.

SUMMARY AND CONCLUSIONS

The security provisions of ASAP are rudimentary, but are relatively simple to understand and use. Since the language is compiled, rather than interpreted, and most of the security features are fully exercised at compilation time, rather than at execution time, the security aspects of the system are not expensive to use. It will be interesting to observe in time whether these simple and inexpensive features of the system will be widely exploited by users. If not, then perhaps security considerations have been overemphasized by those in research and development and do not constitute an important general problem. If they are used, it is likely that users will become more sophisticated in their requirements and these simple classifications may not long suffice. It is, of course, no problem to increase the complexity of record selection conditions, or the number of classifications in either field or action restrictions, but it may turn out that other types of security exist, with concepts unlike those here.

Clearly, the ASAP security system is mainly designed to prevent the casual user from gaining access to information he should not see, and the determined professional would have little trouble going around these security measures, using other languages.

It is interesting to consider the problems of implementing even this rudimentary security system for a conventional file—that is one created, updated, and interrogated by programs written in a contemporary general purpose language (PL/I, COBOL, etc.) and executed under a standard operating system. One might then conclude that the task is most formidable, and that if security is a real problem, the software now in general use is not up to solving it.

The CODASYL Data Base Task Group[8] has proposed language extensions which permit one to specify security and privacy restrictions. They allow for quite general restrictions, and would probably lead to a far more costly implementation of security than that of ASAP. A more general treatment of methods for reducing the cost of information security systems is forthcoming.[9]

REFERENCES

1 L HOFFMAN
Computers and privacy: A survey
Computing Surveys 1 June 1969 pp 85-103

2 W BATES
Security of computer based information systems
Datamation 16 May 1970 pp 60-65

3 C WEISSMAN
Security controls in the ADEPT-50 time sharing system
Proc AFIPS 1969 FJCC pp 119-133

4 H PETERSEN R TURN
System implications of information privacy
Proc AFIPS 1967 SJCC pp 291-300

5 B ALLEN
Danger ahead: Safeguard your computer
Harvard Business Review Nov-Dec 1968 pp 97-101

6 L MOLHO
Hardware aspects of secure computing
Proc AFIPS 1970 SJCC pp 135-142

7 *ASAP system reference manual*
Compuvisor Inc Ithaca NY 1971

8 *CODASYL Data Base Task Group*
October 1969 report

9 R CONWAY W MAXWELL H MORGAN
Security and privacy in information systems: A functional model
Comm ACM 15 4 April 1972

A graphics and information retrieval supervisor for simulators

by JAMES L. PARKER

University of British Columbia
Vancouver, Canada

INTEREST IN SIMULATION

Interest in simulation by computer has grown steadily over the last ten years. This interest has become even more strong in the last couple of years with the advent of popular interest in environmental and social systems. Frequently the most limiting factor in the simu'ation and in the evolution of the techniques of simulation is not the ability to generate the programs to do the arithmetic. It is the ability to handle in an easy way the information flow which results from the simulation and to provide users—particularly those who are not intimately familiar with computers—an easy versatile, hands-on way of displaying this information.

The above need is felt in two ways. First when a new simulator is written a significant portion of the effort of its design and coding goes into that portion of the code which maintains the data base on which the simulator operates. Another significant portion of effort goes into the system which allows the simulator to communicate with a user. This repetitious diversion of energies and money from the primary interest of the people who write simulators (the arithmetic itself) is at this time severely limiting the growth of simulators. In addition the lack of easy graphics display and information retrieval makes it more difficult to correct and evolve the simulators. One can correct and calibrate a simulation more rapidly if one can see exactly what it is doing and if there is a system which allows a sort of throttle control over the activities of the simulator.

The current limited usefulness of simulators can partially be attributed to the fact that the real people who should use the results of the simulators are people who are not familiar with computing equipment. They frequently have only an intuitive feel for what they would like to display and how they would like to see it. Most simulators currently built are left to wither in the academic realm because there is no mechanism for communicating the powers of these simulators to the people who are really interested.

It is the purpose of this paper to explain an information retrieval and graphics supervisory system which will accept most forms of simulation and execute them as subsystems. It provides users with an easy and flexible way of maintaining the data output from the simulators and of displaying this information in graphic and tabular form.

THE IIPS PROJECT (Inter-Institutional Policy Simulator)

The work that is described in this paper was done with a particular simulation project in mind, even though the system itself is not simulator dependent. This project is the Inter-Institutional Policy Simulator project financed by Ford Foundation and as a joint effort between the University of British Columbia, the City of Vancouver, the Greater Vancouver Regional Board, and the Province of British Columbia. The purpose of this project is to involve several diverse institutions in the development of a simulation which can then be presented to the various governmental and social agencies in the Vancouver region. The project is characterised by having several groups working on diverse simulators which must interact. Also its ultimate audience does not in general have a high familiarity with computers. It is also a requirement with this system that it be independent of the types of devices on which output is to be displayed. This project therefore has made an ideal testing ground for the ideas presented in this paper since the variety of people and the variety of simulators involved makes a maximum test of the design of the system itself.

THE SUPERVISOR FROM THE USER'S POINT OF VIEW

This description from the user's point of view will freely use examples based on the regional project being done in the Vancouver area. This means that, because this simulation is basically over time and subregions of the study area, the dimensions of time and region will be used liberally in the following descriptions. As far as the simulator supervisor itself is concerned, time is not a dimension any different from any other dimension or key in the information retrieval system. The only exception to this is the fact that the simulation supervisor does assume that there is one dimension in the simulation which is unique in that the policy interventions explained below are over a certain interval of this dimension. In the vast majority of simulations being built today this dimension is time, so there is no confusion. This dimension could just as easily be distance, however. Let's say that one is simulat:ng a rocket trip from the earth to the moon. The dimension chosen might be distance rather than time. It is conceivable for the simulation of an electronic circuit that this unique dimension would be a voltage level. In the following discussion, however, it will always be assumed that this dimension is time over a subset of regions.

THE INFORMATION BASE

The user of this system sees three things. The primary aspect of this system from his point of view is a large data base which represents all of the available information about the system being simulated over a time interval in years which is chosen by the user. As far as the user is concerned all of the data base information is always present in the computer files. He simply has to display whatever aspects of the system are of interest to him.

The second element which the user sees is called the policy vector. This is a set of variables each of which has a name conveniently chosen for the user. The names themselves may be redefined by the user. Each variable can take on different sets of values over time. As an example, the interest rate prevalent in an economic situation would be a single variable which the user might want to set to different values for different subintervals of the time span in which he is interested. These variables which can be set by the user constitute the policy interventions which the user can make into the model. In this way the user can attempt to study the effect of certain policy decisions and to see the differences in the display for different policy values. Each policy variable has a certain default value at the time the user begins a session with the simulator. He can change this default value to any value he chooses over the entire time interval he is studying or over a subinterval. Assume that the time interval is 1970-1990 and assume that the initial value for the interest rate is 7 percent. The user can change this value over the whole time interval or he can change the value for a subinterval. For example, he could make the interest value 8 percent in the interval from 1980-1985. As far as the user is concerned the current state of the data base reflects the set of policy values which he has chosen and they also reflect the total situation over the whole time interval he is studying. He can also set the time interval that he wants to study.

The third element available to the user is the command language with which he requests displays and asks for changes in policy elements. Although these are the two main functions of the command language there are many other commands associated with the type of displays desired, the devices available, and with saving and restoring portions of the information base.

SOME SIMPLE EXAMPLES

Before describing in more detail the structure of the information base or the command language, it might be well to show a couple of examples of the types of displays a user might request. Let us assume that the user is aware that the figures for population for a set of regions in the study area are available in the data base. Assume that there is one population figure for each region and each year and one total figure called POPULATION for each year. In that case, he might make the following simple display statement.

DISPLAY POPULATION BY TIME

This produces a simple graph shown in Figure 1.

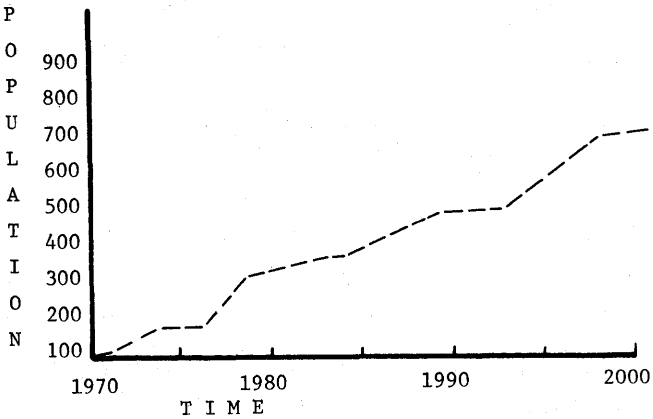


Figure 1—Simple display of population by time

Then let us assume that the user gives the command

CHANGE INTEREST RATE TO 8 PERCENT

He then repeats the first command. In this case, he would get the following graph in Figure 2 displaying the population figures reflecting his policy intervention. As in indicated later it is possible to split whatever device he is using for display into several areas. Using this feature, he could have displayed these two graphs one beside the other on his screen or his paper. He could then compare directly the two population figures. It was also possible for him to give the following command as his second display command and he would have the two graphs superimposed over each other. Thus he could compare them on the same area.

DISPLAY POPULATION BY TIME
SUPERIMPOSED

Or he might have given the following single command which would have produced the same results but with each graph marked by the corresponding interest rate:

DISPLAY POPULATION BY TIME
BY INTEREST RATE FROM 7
PERCENT TO 8 PERCENT

This would have produced the graph in Figure 3.

The simple examples give a clue to the power of the system by showing the direct control the user has over the actions of the simulator. It also suggests that there exists a wide variety of types of displays. The following shows how the user relates these displays to whatever devices he has available and what display types are available.

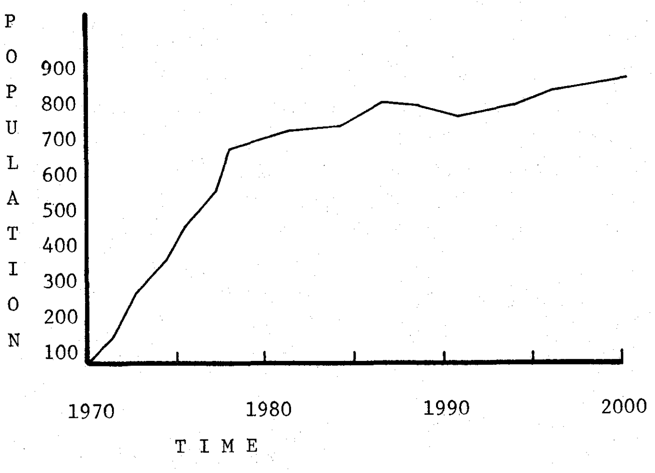


Figure 2—Re-display after policy change

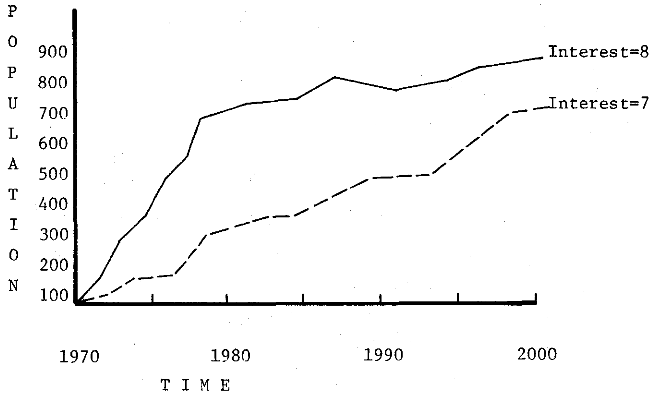


Figure 3—Use of policy intervention in display command

DISPLAY DEVICES

The system is set up so that no matter what sort of device the user is working from he can receive some type of display. The limits of the display for given device type are dependent on the quality of display that the device can support. If the user is sitting at a teletype, he can receive graphs of whatever resolution a typewritten line can support. Similarly, if he has a line printer available to him, he can receive plots on the line printer. If he has a scope available, then he can receive his displays with the much greater resolution and flexibility that a scope allows. Finally if he has a Calcomp, he can receive the same resolution as that of the scope but with the limitations that the use of paper obviously implies. The system as implemented at U. B. C. supports an Adage type non-storage scope, a Techronix scope, Calcomp plotters, line printer, 2260 display, teletype or 2741 disp'ays, and a dot printer.

The user has quite a bit of flexibility in the way in which he can relate his displays to the particular devices. In the above sample command, the defaults were all in effect so that the user did not have to worry about assigning the graphs to areas. For a particular device, the user is allowed to cut the face of the device into 1, 2, or 4 areas. This is most advantageous when using a Calcomp or storage scope. It is also possible that the user has more than one device available at any given time. If he has for example a scope and a Calcomp this allows him to develop certain images on the scope and then have them plotted on the Calcomp. In the case that the user has established multiple areas he must indicate the area on which he expects a display to appear. As was evident in the above examples, the user can cause a new graph to be superimposed on an old graph or he can start with a completely fresh image in an area. It is also possible in the command language for him to move a current display from one area to another. This allows an evolution of a display on a CRT type device and the creation of hard copy from a device like a Calcomp. In addition to this the graphic supervisor is set up so that it provides a memory buffer of images for each area currently in use. This implies that whatever images have been created for a given area, a user can always step back and review a previous image or move forward and again see the current image. The number of images contained in the memory buffer for each area is defaulted to five but it can be changed by an individual user.

RATES OF DISPLAY

It must be remembered that the whole purpose of this system is to display information quickly to a human being and in a form which he finds understandable and pleasing. In light of this it can be observed that a static display is not in general pleasing to a human being. It is much more intriguing and informative to see something that moves. For this reason, the displays on this system have been given a wide flexibility in the way in which they are displayed with respect to time. Naturally these methods of display with respect to time are much more flexible on a CRT but some of these principles are also applicable to paper graph displays.

Each graph is considered to be a subset of a graph with four dimensions. It may seem rather surprising to consider a graph as having four dimensions. However, if one can superimpose several graphs in one area, one already has three dimensions, as in Figure 3. In Figure 3, the three dimensions are population, time and interest rate. Do not confuse time as used here with the idea of the real time of the person using the system. Time here is simply one of the data variables from the information base. If a sequence of figures such as that in Figure 3 could be displayed where each frame in the sequence corresponds to the given value of some fourth variable, the display would effectively be showing the relation of four variables. Not only that, but it would be showing it in a way that would be fairly palatable to human understanding. If the user can see the change in his graphs from one frame to the next, it is actually a fairly good way of transmitting information to a person. The idea of the "rate" in these displays is based on these two ideas—that one can display a four dimensional graph and that the graph can be displayed through time.

THE RATE CLAUSE IN THE DISPLAY COMMAND

The general form of the display commands for graphs is as follows:

```
DISPLAY (SEQUENCED) BY X (SE-
QUENCED) BY Y (SEQUENCED) BY
Z (SEQUENCED) BY T AT RATE R
```

The above example does not include all of the features of the DISPLAY command but includes those which are relevant to the rate of display. It is intended that the SEQUENCED phrase can occur only in one of the four positions. These four positions represent the four dimensions which will appear on the graph. The variables X, Y, Z and T are to be replaced by data names from the data base. The form of the display is determined by which variable has the SEQUENCED phrase associated with it. If the SEQUENCED phrase is associated with the last variable T, then the system merely pauses after each complete frame. Note that it is possible to have less than 3 dimensions in a given display command. The rate R is in tenths of a second. If the word PAUSE is used, then a single frame of the graph is displayed and the next frame is not displayed until the user generates an interrupt on whatever type of terminal he is using. This allows the rate of display to be completely controlled by the user. He can then investigate the characteristics of a single frame before he proceeds to the next frame. Otherwise the image is built up at whatever rate he specifies. The benefit of this type of approach is that it allows one to display more information and to give a person a feeling of something growing. As the graph emerges, one gets the feeling of being able to see a process.

The SEQUENCED phrase can also be associated with any of the earlier dimensions. If it is associated with the Z dimension, then the individual frame of the graph is also built up with pauses in between its parts. Associating the rate with the X dimension also generates a different pattern or growth of the graph and different pattern of pausing. This will be particularly beneficial when using the model for instructional purposes, in which people can discuss a set of policy interventions and then test them out on the model. At this point one can guess what is about to happen in the graph as it grows. After seeing part of the graph one can make predictions as to what will happen next. This will stimulate discussion and the use of the individual's imagination as to what the simulator might do. For example, a policy intervention might require a period of several years in order to take effect. This will allow people to stop and discuss their ideas about how long it might be before the policy takes effect. Since one of the primary purposes of a simulation like this is to stimulate people's imagination, this rate or throttle control on the graph output is very valuable. Figure 4 shows the three different types of growths of graphs which can be displayed. In the succeeding paragraphs, other types of displays than graphs will be explained. Some of these can also have the rate associated with them. At this time no meaning has been associated with the concept of putting the SEQUENCED phrase on the Y dimension.

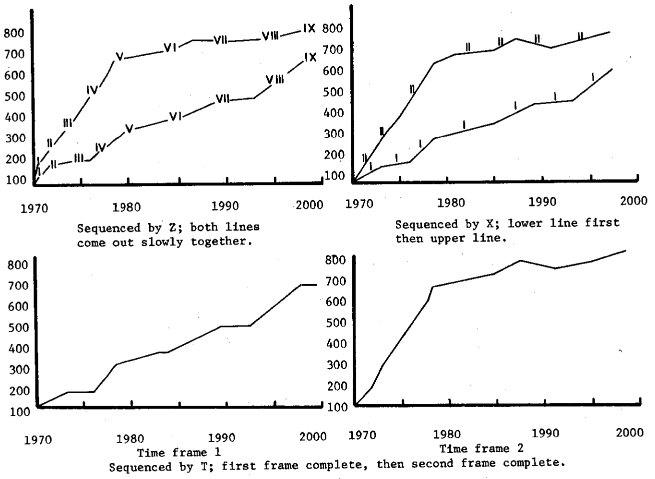
Lines marked with the same number of cross lines are displayed simultaneously. It is difficult to indicate motion as below by printing on a static sheet of paper.

TYPES OF DISPLAYS

There are actually more types of displays available to the user than those indicated above. Any graph may be superimposed on any other. The five types are:

1. Map
2. Point
3. Line
4. Bar
5. Contour

BAR: The bar graph is of a standard form. It is assumed that if more than two dimensions are expressed in the display statement for a bar graph that the third dimension will indicate different time frames. It does not make sense to have more than three dimensions for a bar graph.



Sequenced by Z; both lines come out slowly together.

Sequenced by X; lower line first then upper line.

Time frame 1

Time frame 2

Sequenced by T; first frame complete, then second frame complete.

Figure 4—Different sequences of display using rate with *x*, *z* or *t* axis

LINE: The line drawing is just the standard line graph which has been explained above. This can have from two to four dimensions.

POINT: If the display command is indicated to be of type POINT, the first two dimensions are taken to be the X,Y coordinates of the point. Point data may or may not be superimposed over a map. If the X and Y coordinates are the coordinates of some regions on a map, then it can be superimposed over a map. There are other cases, for example population versus unemployment over several regions, which would generate a scatter diagram for all of the points within the specified time interval and for all of the regions in the model. In a point graph, if the third dimension is given it is assumed to be a Z value and the numerical value is drawn in on the graph at the spot indicated by the X and Y coordinates. If there is a fourth dimension, it is assumed to always represent different time frames in a point graph.

MAP: Maps are generated from files in the data base which are the output of point digitizing equipment. They are currently all outline maps. Any point data can be superimposed over an outline map.

CONTOUR: This type of display is generated from point data with at least three dimensions. In this case the first two dimensions indicate the location of point on the map as in the point graph and the third dimension indicates the value to be used by the contour generating program. If there is a fourth dimension it is assumed to represent different time frames as always.

The form of the display command to indicate the type of display is as follows:

```
DISPLAY/X/BY/Y/BY/Z/BY/T AS MAP
                            BAR
                            LINE
                            POINT
                            CONTOUR
```

See Figure 5.

THE COMMAND LANGUAGE

The above examples have given references to various commands in the language. Although it is not the intention of this document to produce a user's guide to the language, several commands are explained below with their features to give an idea of the range and flexibility of the system.

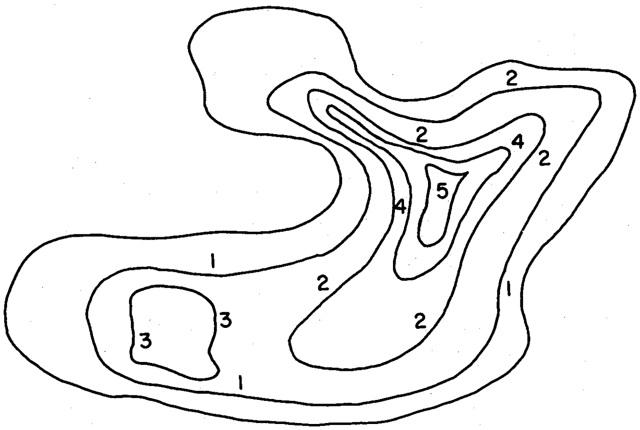


Figure 5—Map with overlaid contour lines. Contour may be formed using any data variable from the data base which has map origins

The commands are broken into three groups: display and data manipulation commands; systems and graph manipulation commands; and user aid and text formatting commands.

The display and data manipulation commands are LIST, DISPLAY, SAVE and RESTORE.

The LIST command has several options. It can list for the user any of the names in the system, such as the names of the policy vector interventions available to the user, the names in the information base, the file names into which the information base is organized, or any text replacement names which the user has defined. The list command can also output data from the information base in tabular form if this is more desirable than having it in graphic form.

The options on the DISPLAY command which have not yet been mentioned are that data may be summarized according to another variable which may or may not be in the display. For example, if population data is present in the information base by region for display purposes it could be summarized by time so that a single value would appear for the total study area for each time interval. Also on any one of the four dimensions, limits can be attached by typing FROM lower limit TO upper limit BY increment, where the increment value is simply the value used to step from the lower limit to the upper limit. Soon there will be added a feature to the system whereby any of the display dimensions can be replaced by an arithmetic expression. Then given unemployment, population and total population available in the data base, one could say 'DISPLAY UNEMPLOYMENT/POPULATION BY TIME,' and this ratio would then be calculated. It is also possible in a display command to introduce a policy vector entry. For example:

DISPLAY POPULATION BY TIME
BY INTEREST RATE FROM 8% TO 9%

It is very likely that the user, after developing a data base and a set of policies in which he may have invested quite a bit of time and money, may want to save all or part of the current status of the system. It is also possible that the user may also want to investigate a decision tree of his own policy strategies and to compare the results. In order to do this he must have the capacity to save all or part of the system in his own private files. This is the function of the SAVE and RESTORE command. The SAVE and RESTORE commands can save the total contents of the system, the contents of any one of the data files, or the symbol names that a user may have defined. See below for an explanation of the user defined symbol names.

SYSTEM AND GRAPH MANIPULATION COMMANDS

These commands set up the structure of a given session using the simulator supervisor and alter the status of the system. The ESTABLISH command initially indicates which devices are available to the system and how many areas each device should be broken into. This allows the use of multiple devices of different types simultaneously. For example, in one location in which the system is used, there is both a storage scope and Calcomp plotter available.

Since each area has a memory buffer, there must be commands for displaying images stored in this buffer. These are the GO BACK and GO FORWARD commands. These commands will also transfer an image from one area to another. This allows one to take a display from the memory buffer of a scope device and to display it on hard copy such as a Calcomp.

The ERASE command simply blanks the image on one area. The primary use comes when the system is being used for teaching purposes and one wants to have the viewer's attention directed elsewhere.

The SET command changes a variety of different system functions and defaults. This is used to set the number of images contained in the memory buffers, certain defaults about the way files are to be handled and so forth.

USER AIDS AND TEXT HANDLING COMMANDS

The EXPLAIN command can be followed by any of a variety of names. If it is a name of an item in the data base, a prewritten explanation of this item will appear. If it is a command name, a portion of the command manual will appear. If an error has occurred and the command says "EXPLAIN ERROR", a prewritten explanation of the error message will be printed out.

The HELP command currently supplies the user with information about where he can obtain various types of information about the system. It is hoped that this function will be greatly expanded in the future.

It is possible for the user to define any character string as any other character string and to have this text replacement occur in his command. The ability to redefine text allows each user to build up during the session the environment in which he operates. This means that he can redefine any name in the system to be a name more comfortable to his own way of thinking. He can also give names to graph descriptions which he feels he wants to display frequently. This will be particularly useful when users begin to generate arithmetic expressions which they do not want to repeatedly type out. The commands here are DEFINE, which defines one string as another string for replacement purposes, i.e.,

```
DEFINE POP AS POPULATION_VALUES
```

This would allow the user to use the characters POP whenever the full name was required. DROP followed by a string simply drops that string from the replacement table.

```
DROP POP
```

It will frequently be the case that a user discovers that he wants to name a whole description of a graph. So it is very convenient to have this built right into the display command itself. The command is then DISPLAY followed by the new name followed by the word AS followed by the usual graph description. This enters the name given in the replacement table and enters as its replacement the string describing graph as below.

```
DISPLAY P_GRAPH AS POPULATION BY
    TIME BY REGION AT RATE 40
```

THE SUPERVISOR FROM THE SIMULATORS POINT OF VIEW

One of the primary goals of this system is to make it very easy to add a new simulator and to minimize the burden of input/output command parsing and display on these simulators. For this reason the world into which the simulator fits is made extremely simple. As far as the simulator is concerned, the communication with the outside world consists of reading variables from files which are accessed by a file number and writing other numbers into other files. In case of an error there is a routine which is called with an error message number and which has some macro capabilities. The files that the simulator sees are basically sequential files with keys allowing either a sequential or random read or write. The file descriptions which define the names of the values which will be put in these values are written out in simple character format very much like the data description for any high level language. These file formats are shared by all the simulators in the system at one time, so that one simulator can read the files generated by another simulator. Since these file descriptions are in character format, they can be easily changed and evolved as the simulators themselves evolve. It is only required that (as in FORTRAN style input and output) the simulator assumes the function of each variable in each row that it reads in from the file. For one given file, the simulator must be aware when it is written that the fourth variable of this file is population.

MORE DETAIL ON THE INFORMATION RETRIEVAL SYSTEM

The information retrieval system is really the heart of the simulator supervisor. It must be able to access the files by key and it must be able to support a fairly complicated key structure within the files. For example, there will be files which are indexed by time and region and there will be some files which will be indexed only by region. Yet for display it would be necessary to relate these two files by region alone ignoring the time key on one of the files. There would also be files by time and region and other files by time region, and time of day. This effectively means that the information system must support a hierarchical data structure with multiple keys. However, the vast majority of accesses to the data base by the simulators are made in a completely sequential fashion. That is, very little of the structure is used in a hierarchical fashion. The possibility must be there when it is required.

In order to maintain this facility, the information retrieval system has been structured to operate on sequential files with fixed length records. These are designed to look like FORTRAN vectors; they are just a row of variables. In order to maintain the facility of multiple keys, each file may have several keys which are concatenated together to form a single key. If one wants to generate a facility for accessing the kind of structure mentioned above where one file is by time and another by time and region, the implementation must have two files with the information recorded by key in each file. To do this key lists are kept for each file which gives the key value and a line number or record number for locating the corresponding record.

Since it is possible to change policy variables over certain intervals of time it is also necessary that there be a mechanism for invalidating portions of the data base. For example, if the current data base had been computed from 1970 to 1990 and the interest rate were changed in 1980, it would be assumed that some of the output from some of the simulators would no longer be correct if they were influenced by the interest rate. Therefore those files would have to be erased beginning with 1980. This can be accomplished by simply erasing the elements in the key list which have year values after 1980.

This organization of the information retrieval system allows very simple access routines for the simulator. Yet it will allow a fairly complex hierarchical structure of keys if necessary. The fact that the user assumes all the data to be present at a given time and that the data is actually computed by the simulators only when it is requested for a display means that unnecessary computations never have to be made. Therefore the information retrieval system must be able to communicate to the supervisor what data is actually present in a file at a given time.

In order to explain to the system the structure of the files, and the interdependencies among the simulators for data, there must be a set of simulator descriptions and the file descriptions. These file descriptions as mentioned above are simply lists of the variable names that will be used by the user for referencing the data. It also includes other characteristics of the variables such as the maximum value that they should be allowed to obtain and the minimum. For each file there is a description of the simulator which writes the file and how to load that simulator in order to execute it. There is also a set of simulator descriptions which describe the files each simulator reads. This is sufficient information for the supervisor to decide on a given display command what data is required, what data is missing, which simulators must be called to generate the data and in what order the simulators must be called. The rule is that a single file may be written by only one simulator. This is required so that the supervisor can tell how to replace data which has been invalidated by policy changes or which may never have been computed in the first place. In order to generate missing data, the supervisor steps through the years in the current time interval and at each year runs the simulators required to generate data for that year. These simulators are dynamically loaded and are kept in core as long as possible until another simulator is required to run.

AN OVERALL FLOW DIAGRAM OF THE SYSTEM

The system is broken up into three major sections of code. The primary section is the supervisor itself. This code does the initialization work of reading in all the tables, reads in each command from the user, and converts the alphabetic strings in the command to integers which are merely reference numbers corresponding to the strings. This allows command interpretation to be done very easily. It is at this point that text replacement is also done. The commands then go through a large switch to individual subsections of the supervisor which process each command. The second section of coding are the I/O routines. These routines maintain the key lists for the individual files and do all of the sequential and random reading and writing requested either by the simulator or by the supervisor. At the time the simulator originally begins running it is assumed that each file that is non-empty is in a read-only master copy. As long as display commands are generated for these files this read-only copy is used. As soon as simulator wants to write a file a scratch copy is made. This copy can either default to a temporary file or can be put into a private file of the user, depending on a systems switch. Notice that the I/O routines, except for saving original data, treat observed data and simulator output exactly the same. There is no distinction between data bases which were

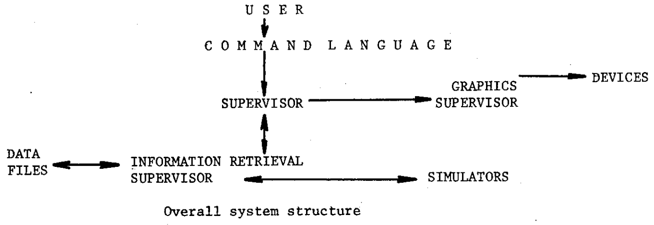
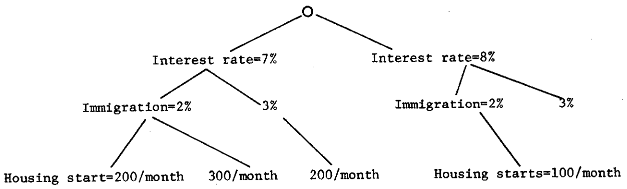


Figure 6—Overall system structure

originally observed and which are modified by simulators, data bases which were written only by simulators, and data bases which were only observed. This symmetry of treatment coupled with the ability to describe files in format-like statements allows the structure of the files to evolve very easily.

The third piece of code involved is the graphics supervisor. This section of code maintains the current state of all of the display devices in the system. It also maintains the image buffers, the memories for each area of each display. It is the responsibility of this section of code to convert a standardized graph description which it receives from the supervisor into the proper display for each device in the system. It is also the responsibility of this section of code to OR the graph images together into single images when superposition has been requested. This part of the program also does the computations required for contouring and displaying of bar graphs and so forth.



```
DISPLAY POPULATION BY TIME
DISPLAY GEOGRAPHICALLY SINGLE_FAMILY_DWELLINGS AS CONTOUR
DISPLAY GEOGRAPHICALLY POPULATION SEQUENCED BY TIME AS CONTOUR
```

At each level display might be made of population, land use contours, etc.

Figure 7—A possible decision tree for investigating policy alternatives

EXTENSIONS

The most exciting extensions to this system will be in the area of providing facilities to the user for developing policy strategies. The first cut in this area will be to give him the ability to develop a decision tree of alternatives, and to compare and display data across these different alternatives. See Figure 7. At this point it would also be of value to be able to provide him an analysis of which variables in the data base are affected and at what level by changes in the policy variables. At first this can be done on a guesswork basis by knowing which data is invalidated by changes in which policy variables. Later on, however, it may be interesting to do more in-depth analysis by actually running the simulators for a short period, noting which variables change, and attempting to give some indication of the derivatives involved.

What would also be very interesting in this context is to generate a rudimentary artificial intelligence device which searches out policies which tend to satisfy certain goals set up by a user. In this case the interaction between the user's intuition in suggesting goals and a rudimentary artificial intelligence device might produce some very exciting results.

CONCLUSION

It is strongly believed that this system will bring the power of large interactive computer systems rapidly to users in two different ways. Primarily, it will provide an ability for an untrained user to investigate in great detail and according to his own experience the workings of a set of simulators. Secondly, by relieving the simulator writers of the burden of providing user interface and information retrieval it will allow the quality of the simulators and the types of the simulators themselves to be evolved very rapidly.

NASDAQ—A user-driven, real-time transaction system

by NAT MILLS

Bunker Ramo
Trumbull, Connecticut

INTRODUCTION

By Christmas time of 1970, 1200 brokers active in the Over-The-Counter securities market nation-wide, had had a new Bunker Tamo CRT terminal placed upon their desk. Through these "NASDAQ" terminals, they would be expected to watch and change all their quotations from then on. Within a month, they were confidently using the data in the system, that they were themselves updating, and turning their backs on almost all of the previous phone checking and "pink sheets."

By March, 1971, Barron's magazine had reacted to NASDAQ as follows: "To the amazement of virtually everyone, the complex network of cables, computers and cathode ray tubes . . . breezed through its shakedown period with scarcely a hitch. What's surprising (is) . . . the almost instantaneous success and acceptance NASDAQ has scored."

And before the new year was half gone, the talk had turned to the feasibility of automating the exchanges themselves. NASDAQ was becoming a model of how aptly a computer complex could serve a national trading community.

"BEATING THE ODDS" IN REAL-TIME SYSTEM CREATION

The NASDAQ system is but one of many large real-time, transaction processing configurations that have been attempted during the past 10 years and in most cases been completed, sooner or later, during the past 5 years. Many of our computing fraternity (and sorority) have been involved in these efforts, and many more will be immersed in similar efforts during the next 10 years. In this paper, we are offering the highlights of our own particular experience, and some of the reasoning behind the course we took, which did "beat the odds."

In fact, evaluating the design decisions made in building the NASDAQ system might well be viewed as one half of a very useful object lesson in the problems and pitfalls to be met in developing large on-line, real-time systems. The other half of this object lesson would then be the comparative evaluation of the design options chosen in less fortunate projects where the pit was fallen into. It would be presumptuous, and certainly not fitting for us to make a critical analysis of such projects. Therefore, we confine ourselves to presenting the following picture of our own effort.

THE SYSTEM IN PROFILE

The NASDAQ System is a nation-wide, real-time configuration now serving the quotation needs of the Over-The-Counter securities market. NASDAQ stands for the **N**ational **A**ssociation of **S**ecurities **D**ealers **A**utomated **Q**uotation system. It is centered in a pair of Univac 1108 multi-processing computers in Trumbull, Connecticut (see Figure 1). Radiating from this center is a network of data lines to Bunker Ramo Model 2217 CRT's in the individual offices of sub-

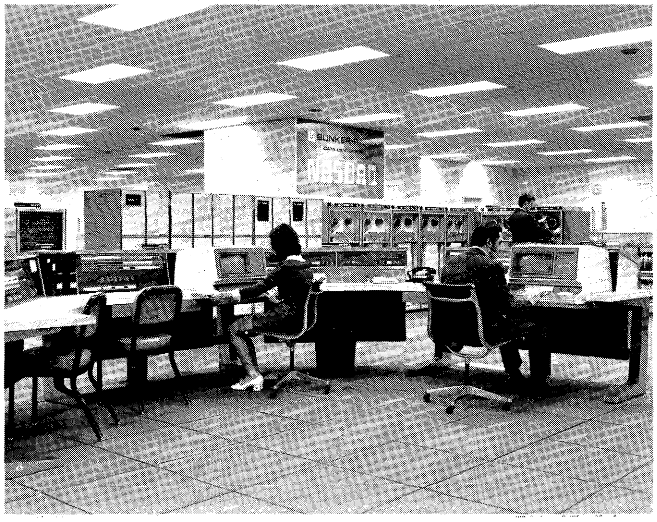
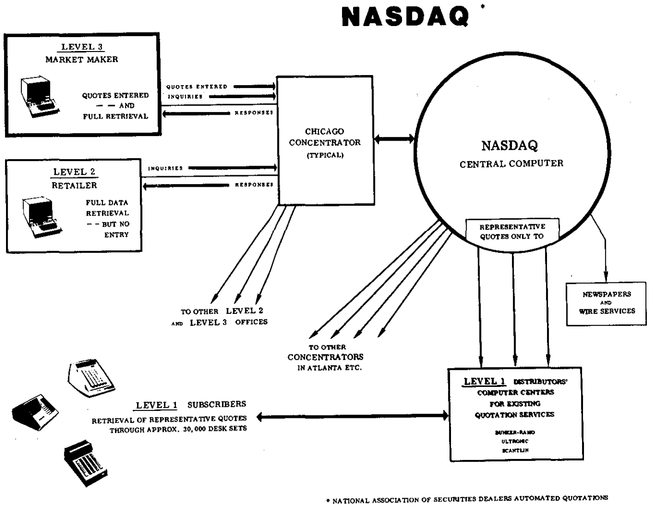


Figure 1



* NATIONAL ASSOCIATION OF SECURITIES DEALERS AUTOMATED QUOTATIONS

Figure 2

scribers. Controlling and concentrating the traffic in this network at four regional points are duplicate Honeywell DDP-516 computers, each with a Bunker Ramo designed Communication Control Unit as a front end interface (see Figure 2). In each office being served is at least one Over-The-Counter Control Unit (OCU) to support the terminals installed for NASDAQ (see Figure 3).

At the Trumbull Center, as units integral to the central processor, are a group of auxiliary memory devices and communication traffic handlers. In the hierarchical levels of the directly accessible storage is held the on-line data base, from which quotation, index, volume and other information is retrieved for remote display. The system restricts to a particular level of terminals in the offices of market makers and the NASD itself the functions of data base updating and control.

Activity is, as of this writing, averaging over 900,000 calls per day, and peaking at busy periods at over 110 calls processed per second. The size and scope of the system is still expanding. At this point, over 900 offices are tied into the system, with 1,400 terminals installed.

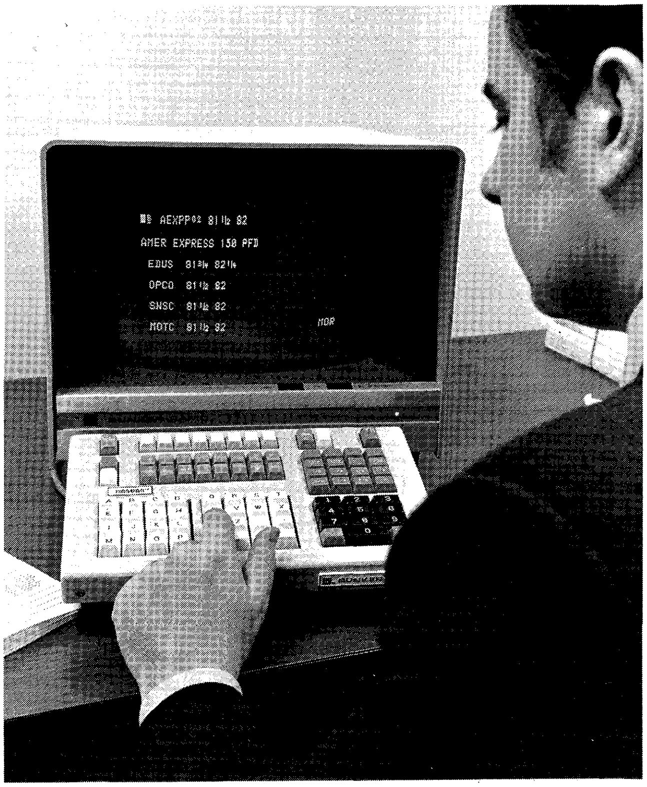
The data base is built around records for over 2,800 securities, active in the system on the basis of over 19,000 quotations by market makers taking positions in them. The volume of purchases and sales is being reported into the system daily and now exceeds that of all other exchanges combined, except the NYSE.

THE UNIQUE CHALLENGE FACING IMPLEMENTATION

Of course, this whole system and its functions did not arise haphazardly. The start of the design effort was the recognition that the implementor of NASDAQ faced a formidable task for several pivotal reasons.

- The direct users of the system would be brokers interacting amid their hectic daily trading. They would be entering, verifying, and retrieving the data upon which the data base itself would be built. No trained clerical personnel or computer operators could intervene.
- The whole process being automated would actually create an entirely new way of doing business for the Over-The-Counter Market. No precedents or earlier system existed.
- To support a viable market, the whole trading community, nation-wide, had to be on-line together at system start-up.
- A substantial portion of the expected users were reluctant and skeptical about being supported by automation. Any failures or ambiguities in function could be expected to be exploited to question the whole system's utility.

Recognizing the seriousness of these challenges, the system analysis effort began in earnest.



```
AEXPP02 81½ 82
AMER EXPRESS 150 PFD
 EDUS  81¾ 82¼
 OPCO  81½ 82
 SNSC  81½ 82
 MOTC  81½ 82                MOR
```

Figure 3

ESTIMATES OF TRAFFIC LOAD AND MIX

In the effort to provide a solid basis for all design decisions, there were studies in the field, simulations, and functional analyses of many aspects of the processes NASDAQ would automate. Particularly, in order to properly select and locate components of the communication network, we needed informed projections of traffic loads, geographic distribution, and mix of entry types.

With the decision already made to put the central site in Trumbull, Connecticut, the heaviest volume of traffic would obviously be between there and downtown New York City. Other traffic paths were mapped out and studied for potential service demand. Then formal network analysis was applied to the data, using "COPE," Bunker Ramo's own computerized analysis tool.

From the functional aspect, observation of O.T.C. brokers' activities indicated an approximate ratio of seven to one between the number of requests for the quotations in a security and the number of updates of those quotes. The human factors of location, style of work and data interaction were studied in the actual trading environment to obtain further data for the design decisions, which are described in the following sections.

THE CENTRAL SITE CONFIGURATION

For the central processor in the NASDAQ configuration, the Univac 1108 was chosen on the basis of its already proven reliability and the elements of its design which expedited real-time operation. Two units, as a multi-processing complex, were recognized as essential to carry the system's total processing load, though unit processing, when necessary, would be sufficient to maintain all on-line functions.

The total complex is controlled as a configuration by an Availability Control Unit (ACU), which allocates facilities in the multi-processing environment (see Figure 4).

Since swift accessing of a large data base would be critical, an hierarchy of random access storage devices was chosen from those in the Univac line. In this case, a wide range of high-speed and low-speed drums and common core has provided an adequate supply of random access storage for the data base, programs, tables, and dictionaries as needed. From these devices, individual record retrievals, program overlays or even remapping core has functioned with speed sufficient to keep within the specified response time on all transactions.

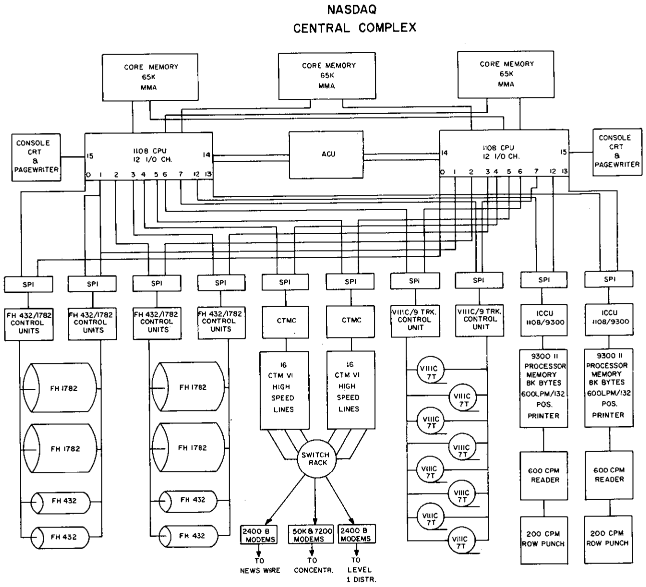


Figure 4

Auxiliary to the central complex is a normal complement of magnetic tape drives, consoles, two Univac 9300 processors driving printers, and card units. For environmental control, dual air conditioning systems have been installed. To assure continuous power, an Uninterrupted Power Supply (UPS) of batteries and a gas turbine motor stands by in permanent readiness.

At the center's terminus of the communication network, a Bell System exchange is installed on site, dedicated solely to handling NASDAQ traffic. Interfaced with the central processors are a pair of communication terminal module controllers (CTMC's) each with its own set of 16 communication terminal modules (CTM's) cross-switchable for carrying all traffic through the appropriate modems to the users via the regional concentrators and to the auxiliary services tapping the data base, such as the public quotation services and newswire distributions.

The upper limit for traffic through this present central configuration was pegged by original design estimates at 220 calls per second. However, built into the hardware plan (and the software design as well) is ample provision for expansion to handle growth in load, and changes or extensions in function.

THE DECISION TO INSTALL REGIONAL CONCENTRATORS

Those who have been close to the NASDAQ development effort from the start feel that the most important

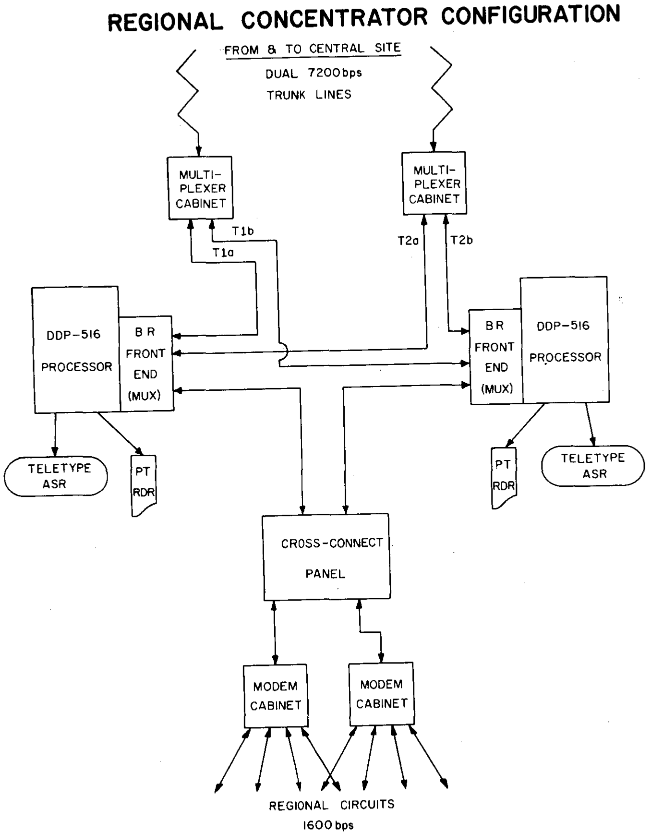


Figure 5

single design decision was that made to set-up regional concentrators for communication processing and polling. The danger of over-burdening the central processing complex with the details of terminal servicing was recognized early.

The network analysis indicated the economic benefits of concentrator placement at New York City, Chicago, San Francisco, and Atlanta. This divided the communication network of leased lines into two levels:

- dual trunk lines transmitting point-to-point between the central site and the concentrators synchronously at 4800 bps. (To and from New York City at 50 K bps.)
- regional circuits between the concentrators, and the control units in the brokers' offices, transmitting asynchronously at 1600 bps, as private line multi-point duplex data circuits. (Asynchronous in order to match the most suitable modem and the chosen line speed.)

At each regional concentrator site, the following equipment was installed (see Figure 5):

- A pair of unitary configurations each backing up the other with:
  —A Honeywell DDP-516 processor (16K Core)
  —A Bunker Ramo designed and built communication front-end interface
  —A paper tape reader
  —A teletype Model 35 ASR
- Connecting this pair of units with its regional lines are:
  —a cross-connect panel
  —two Multiplexer Control Cabinets
  —two Modem cabinets

With this concentrator structure, it was possible to introduce priority handling and line controls at the regional level. For instance, full duplex use of the regional circuits became effective with the introduction of nested polling initiated by the concentrator. With the multi-station local control units all riding their assigned regional circuits, polls could be nested into individual messages and be recognized by the proper addresses at a great saving in traffic load and message turn-around time. This technique also tends to equalize the service for all terminals, by giving priority in the polling process to control units in offices with multiple CRT's.

AUTOMATING THE OTC TRADER'S FUNCTIONS

An often somewhat neglected aspect of system design, the human interface, had to be studied carefully,

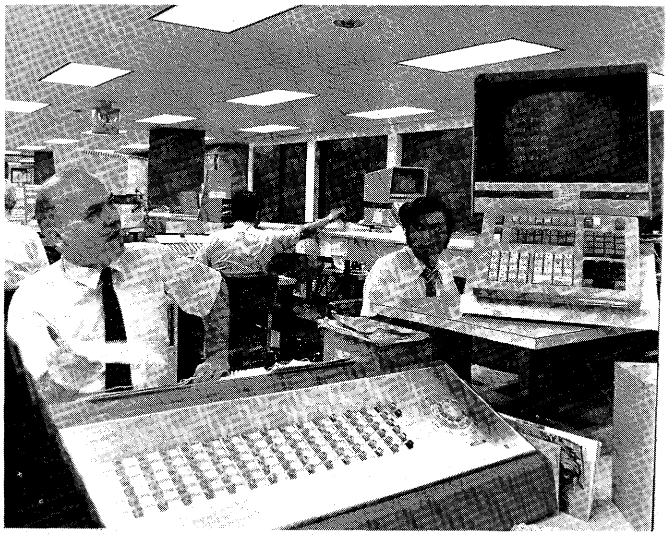


Figure 6

and be coped with realistically for this project to get off the ground. The extraordinarily hectic environment of the trading room of a busy brokerage house has to be seen and heard to be believed (see Figure 6—an unbusy example).

Of course, Bunker Ramo has had some experience in this industry, dating back to 1929 for the first automatic dissemination of quotations. But the direct input of data by the broker at his desk was something entirely new, and a somewhat frightening prospect for seasoned real-time computer professionals.

For NASDAQ, this meant first of all the keyboard layout, and the ease of entry message generation. Thus, a unique keyboard was designed from the table up for this new type of user (see Figure 7), which attempted to support his manner of operation, and to avoid appearing overloaded and baffling. Special function keys directly expedite the setting up of certain messages. In particular, the broker maintaining a quotation in the system is able to change it rapidly with a few keystrokes, or with little more than a directional character (↑ or ↓) to cause the normal change of ⅛th of a point. Other design advances contributed to an overall layout which sought to give the user a "congenial" keyboard. It seems to have succeeded.

The second terminal area to which human factors concerns were applied was the query/response display content. Actually the processing logic of the whole system became query/response oriented by conscious decision. No demand to respond to computer processes is put upon the user. The entries the user transmits are accepted as a whole and displayed as updated for the user to verify or are rejected with an explanatory message, which does provide some self-teaching interaction. The effect has been to enable the users to get what they want from the system with next to no training.

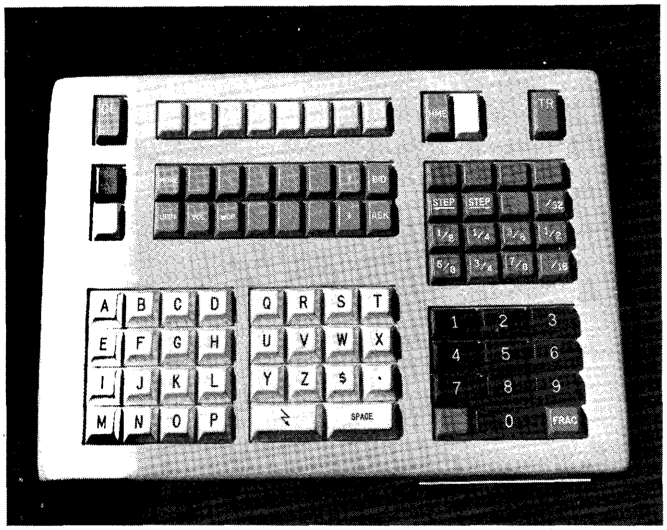


Figure 7

Just behind the individual terminal, supporting up to 24 of them per office, is the O.T.C. Control Unit (OCU). This unit handles the refresh, character generator and control logic for the CRT's. It also provides some by-product functions that assist the user. For instance, it generates a unique office and device address, which it prefixes to every out-going message, thus ensuring protection to the firm's quotations, and enabling office oriented services to be selectively implemented by software processes.

As an example of the flexibility of the OCU, there was the necessity to cope with a late addition to the functional requirements as originally specified. Some brokers wanted to be able to capture the changes in their own quotations as they were entered into a NASDAQ terminal, and post them on automated display boards in their own offices. This was implemented for such users by setting a special bit in the response which would signal the OCU, and by supplying a "Local Post" key on the keyboard to permit such local transmission when desired for non-NASDAQ securities.

This then completes a capsule description of the major hardware elements at each level of the system. It has proved to be a smoothly functioning, coherent complex. The next sections of this overview deal with the program structure which made the system function.

SOFTWARE ELEMENTS

The major NASDAQ components in the software category are the following:

- An expanded version of Univac's "EXEC 8" operating system, supporting multi-thread, real-time transaction processing with queued file accessing and updating. Bunker Ramo extensions to this system upgraded its real-time communication control features.
- The necessary set of on-line functional application programs, processing the full range of acceptable queries to, and required outputs from, the system.
- A hierarchy of recovery programs, designed to reallocate system elements, to restart, or to drop to degraded service as necessitated by conditions.
- A series of off-line processing programs to handle file maintenance tasks, report production, and pre-processing to optimize the file structure of the next day's on-line operations.

- A communications control program to function in each of the DDP-516's of the remote concentrator configurations for:

  —polling each control unit and servicing each regional circuit
  —store-and-forwarding message traffic
  —managing buffer resources and time clock functions
  —printing out (at NYC) the newspaper transmissions
  —monitoring circuits for error rates which might predict line deterioration, and alerting operations personnel to the need for intervention.

- An overriding time-monitoring program to initiate at pre-set times various activities such as:

  —opening or closing all active securities
  —calculating index values
  —disseminating data to news media
  —halting the volume reporting function

- A many-faceted program providing supervisory access of several types to the system and its data as follows:

  —on-line entries to cause immediate display or alteration of file contents
  —on-line intervention to change system parameters or time functions
  —on-line entry of file maintenance transactions for off-line processing
  —off-line batched transactions for normal file maintenance

CREATING A CLEAN MESH OF SOFTWARE

Developing all the elements of the total software environment required a staggering amount of coordination and attention to detail. The general principles followed were the standard ones for real-time programming such as:

—reentrant and reusable code
—dynamic buffering
—program overlays
—record lock-out during update
—automatic system recovery and restart
—queued file accessing within multi-thread logic
—multi-processing utilization of dual processors

On top of taking full advantage of the current "state of the real-time art," certain solutions to problems unique to this system deserve mention.

1. The analysis of expected traffic had highlighted the predominance of quote requests, and among them, for the "top of the list" for any security, that is, what would be the first quote frame on the CRT. The rest of the list of quotations is obtained frame by frame as a "MORE" key is pressed. To save processing time with this first frame frequency, it was decided to keep it always pre-formatted in the security record ready for immediate transmission.
2. In a similar effort to expedite retrieval response, the statistics of accesses for each security are used daily to order the next day's allocating of the address keys of the most popular securities to a core-resident dictionary, and all others to a drum-resident dictionary.
3. The security of the data base was guaranteed by the storage of two copies of all on-line files in cross-drum and cross-channel locations so that the availability of any record would not be affected by the failure of any one hardware component.
4. Buffers are grouped in different sizes and requests are queued first by buffer size needed, and then shifted away from the longer queues, if possible, to the queues of the buffers next in size. Access to a drum record is also shifted to that copy of the record which resides on the sub-system with the shorter queue.

The net result of these and many other techniques has been a quite unflappable system. The meshing of all the programmed modules was accomplished before start-up with the usual around-the-clock effort, but fortunately with a minimum of unpleasant surprises and crash patching sessions.

PUSHING EXEC 8 OVER THE HUMP TO ACCEPTABILITY

Even though the operating system for the Univac 1108 had been around for a while, it was clear fairly early to the NASDAQ implementors that it had its weaknesses, and some serious gaps for real-time system control. Therefore, a special effort was directed toward identifying all the problems that turned up, and all potential problem areas as early as possible.

With alacrity and concentration, the Univac program development staff in St. Paul, Minnesota went to work and produced the necessary enhancements. Some of the improvements, however, were developed by the NASDAQ crew itself, and most of these were then adopted as part of official versions of Exec 8.

The most significant modifications and enhancements to Exec 8 were the following:

—Protection against improper mass storage assignment.

—Ability to optimize drum utilization by using absolute addressing to allocate or access mass storage.

—Ability to bring up or shut down I/O paths and units initially and dynamically.

—The aborting of I/O queues at abnormal program termination to facilitate faster reload.

—Provision for extensive logging and accounting of I/O error events.

—Ability to get "immediate functions" performed by the Exec.

—Revising the drum read and write routines to shift from one in two interleaving up to one for one interface between drum and core. This particular stratagem significantly improved data transfer speeds. It had not been regarded as feasible for the hardware involved, but the step has proven to be sound.

—Ability to reconfigure any component at any time with or without manual intervention.

—Applying the processor monitoring capabilities of the Availability Control Unit (ACU) in order to detect CPU malfunctions.

—Utilizing automatic reboot, whenever possible, to reduce the amount of operator intervention required.

—An automatic dump to high speed drum when the system has to reboot.

—The detection of a rebooting loop and signaling for operator intervention.

—The monitoring for, and the correction of a real-time error loop by automatic program reload.

—Maintaining a Master Configuration Table in core and on drum, containing the current status of the system.

—Ability for an application program to be reloaded without losing its facilities allocations.

—Entry point to Master Configuration Table to enable application programs to obtain necessary recovery data.

—A variety of enhancements of the operator interface with application programs, including a display of the program mix functioning at the time.

—A repertoire of debugging aids such as core and drum traps, dynamic core and drum dumps, and a trace routine.

—The development of a "moving picture" of core displayed on the console CRT. This is a little gem of a monitor routine for debugging and analysis work. For instance, the rhythm and pattern of dynamic buffer allocation could be observed as its core locations changed on the display screen.

—A package of test routines to shake down new levels of the Exec itself.

—A variety of lesser improvements to clean-up and remove unnecessary code, and generally to reduce overhead.

It is not too much to say that this effort was as crucial as any to the success of the total NASDAQ implementation.

THE TACTICS OF IMPLEMENTATION

The NASDAQ project got under way with the contract signing in December, 1968 between the NASD and the Bunker Ramo Corporation. The completion date specified in that contract was two years later to the day.

Many things had to be set into motion. Hardware units had to be designed, checked out and scheduled for assembly line production. A new building had to be planned and built for the central processor site. Systems design, network analysis, and computer selection all had to be tackled in short order. The early installation of the central processors was a big help to all concerned.

One of the things that was not set into motion was the functional specification after it was carefully elaborated and clarified. Of necessity, it was frozen early and fortunately was able to stay that way almost inviolate.

A relatively small crew of engineers, system analysts, and programmers (approx. 50 in all) was assembled, mostly from within Bunker Ramo, to tackle the direct system creation. They turned out to be an extraordinarily dedicated and resourceful crew. Just before Christmas in 1970 they offered a functioning NASDAQ to the waiting world. It had arrived only four days late, and it stayed arrived.

THE PARCEL OF SUBSIDIARY FUNCTIONS

As in most such systems, a variety of subsidiary functions were included in the functional specification, and then turned out to require a major part of the effort. These bonus components, all of which were safely on board when NASDAQ started functioning included the following:

—A median quote, called the Representative Bid/Ask (RBA), for display on the top line of every quote frame, and for public dissemination.
—A set of indices for the OTC market, based on all the stocks in the system, for retrieval and dissemination.
—The display of news bulletins in response to requests from any terminal.
—The provision of direct transmission of RBA's and Indices to the quotation and newswire services.
—A full gamut of overnight off-line processing to prepare the data base for the next day with all file changes made, to produce regulatory reports for the NASD, and to provide statistics on system behavior.
—The support of all varieties of supervisory intervention and file adjustments entered on-line, with some requiring action immediately on-line and some held for off-line processing.

This last area generated the largest single module of programming and required extensive rearrangements in the overlay logic. These areas as a whole required more extensive testing than all the rest of the system combined. In fact, the last critical hurdle before start-up was the satisfactory completion of that part of the cycle which went through the off-line processing from the end of one day into the start of the next day.

THE TEST TO END ALL TESTINESS

To guarantee that the NASD and all its user-members would find nothing in the system to get upset about, a frontal attack on all aspects of the system was mapped out. The functional, supervisory and communication functions in all their complexity required exhaustive testing. The system's capacity also needed testing under over-loaded traffic conditions.

How could we throw every possible permutation and combination of query at the system and verify that we were getting the correct, prescribed response? The type and content of the response in turn depended on a multi-dimensional matrix of conditions, such as the brokers' type, the status of the security affected, the time of day, and the other quotations already entered. In other words, to predict any particular query/response pair, we would have to establish the content of the data base at that point in time for that particular interaction.

Only then could we expect to isolate those responses which might be proper in form but incorrect in content.

The immensity and crucial role of this stage of the system's development forced us to become the midwife of some unique methods to meet the necessity. We realized from the start that we would have to automate the process in some fashion in order to avoid the horrendous, error-prone chore of creating exhaustive (and accurate) test data input.

The possibility of writing a program to generate a full variety of test input in query form, without responses, proved on examination to demand an extraordinary programming effort, among other drawbacks. Even to check response messages on a print-out by eye, or on a dump by programmed comparison were each clearly prohibitive prospects. There had to be a better, yet absolutely thorough method of automating the check-out of system responses.

THE SOLUTION TO OUR PROBLEM

Since 99 percent of the live input data would be entered via the CRT keyboards, and most of the output would be displayed on the CRT screens, these terminals and their control units were the critical message-formatting modules of the system. A processable query and a displayable response would have to be sent from, and be returned to, this control unit.

The accuracy of the *visual* content of keyed entries could be checked on the display before being sent, and the content of the resultant response from the system could be *visually* verified in the same way.

Could we hold this query/response pair in their message formats and then, if correct as displayed, capture the paired record on tape, or if not, divert it for diagnosis? In the second case, the encoded messages would provide precise evidence for remedial action by either programming, systems, or hardware design staffs.

The equipment to handle this task was already at hand in the form of two communication concentrators located in the Trumbull Data Center, adjacent to the central processing unit.

Thus for the test, one of these concentrators performed in its actual NASDAQ role, but as the only *active* front end for the whole system. Thus the system configuration and its software did not need to be altered in any way for test purposes. As far as the NASDAQ system could tell, it was receiving and responding to

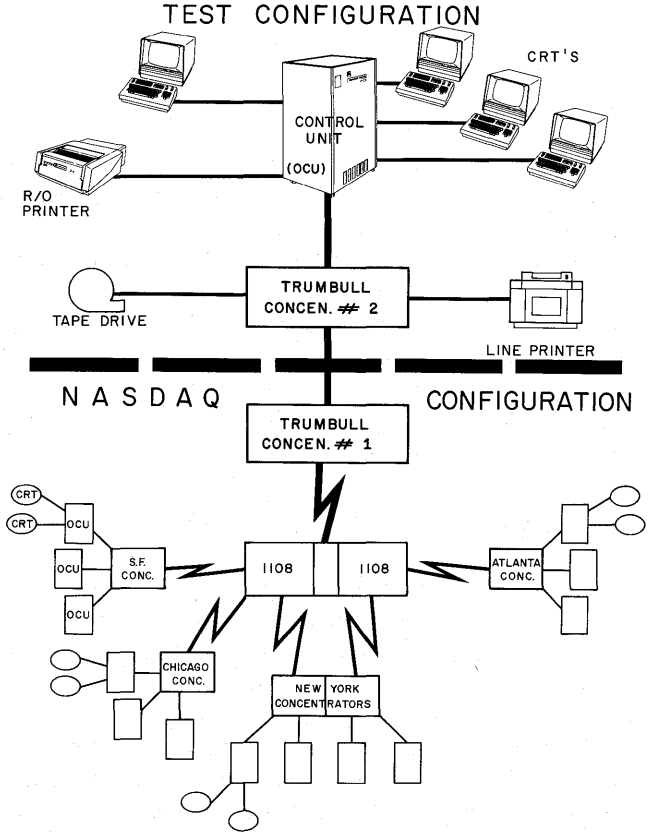


Figure 8

genuine activity from real users (see Figure 8 for a schematic of NASDAQ plus our test configuration).

The second concentrator was programmed as a test monitor and manipulator. This concentrator had to function in a two-sided fashion, appearing to the first concentrator and the central processors just like any one of many possible control units with terminals. On its other side, interacting with the test control units, it behaved as if it were the NASDAQ system itself. Among other facilities, this enabled us to circumvent the restrictive integrity of the NASDAQ design, which accepts file updates only when uniquely identified by the user's own hardware-supplied address.

For test manipulations in general, this programmed concentrator gave us the flexible control we needed. Its program responded to a set of control level entries from the test CRT terminals to trigger the verification capture, numbering, and printing, etc., of the test query/response pairs as created.

Early in the game we decided that we would have to simulate at least six days of system operation. This became necessary not only to allow for some gradation in complexity, but also in order to permit certain global conditions to be amply tested, such as a "delayed opening" of the market on a particular day.

We were equipped to map into core memory and drum files a check-pointed or start-of-day on-line data base which recreated a previous stopping point. Using this facility, we could push ahead with the testing effort in several different areas in parallel fashion. Otherwise with a linear check-out path, each bug would have hung us up until the necessary program patching was accomplished.

By allocating a different set of simulated securities to each major function, we were also able to work around any buggy area until it was ready to be tested again. Within each of these allocations, a spectrum of security types and market maker positions was developed in order to exercise every conceivable variation.

In addition, we set up separate charts to keep track of every scripted entry that generated a line on one of the required off-line, "overnight" reports.

CARRYING OUT THE TEST

Anyone who has ever been involved with the demonstration of an on-line process knows what happens next. With everyone crowded around to watch, the previously infallible gear or program begins to fall apart with a spectacular display of recalcitrance. Well so it went. We set the stage, everyone held their breath, and then the first query we keyed in proceeded to pull down the whole software structure.

Of course, with only six weeks to go before cutover, there was consternation, pandemonium, and dire forebodings from all sides. Scrutinizing the wreckage, the programming team quickly spotted and repaired the fatal flaw. It turned out to be precisely the type of problem that live volume testing or actual use would eventually have hung up on. The unique feature of that first frame was simply the use, in a particular function, of a security whose initial letter was beyond "S" in the alphabet. Our test scripting fortunately had the range of detailed variation to catch this situation.

When all the smoke had cleared, we resumed our testing effort and this time the scene settled into a more normal process.

The check-out continued right up to start-up time, and helped assure everyone that all was well with NASDAQ.

ACHIEVING RELIABILITY

The reliability of the NASDAQ system is by now an accepted fact of life for the OTC market. System performance has been on an uninterrupted basis for 99.92 percent of the scheduled time since January 1, 1971 (total down time of one hour and 57 minutes in 2,376 hours of operation).

Reliability is, of course, critical to the success of any such system after it becomes the functioning structure of a nation-wide market place. Its realization in the case of NASDAQ is the result of a web of carefully designed features.

On the hardware side, it has been due first of all, to the inclusion of units of already proven dependability in service. The long evolution of Bunker Ramo's on-line configurations for the brokerage industry now guarantees the production of terminals, control units and multiplexers which inherit a built-in reliability.

Bell Telephone lines are another matter, but in this area also, Bunker Ramo experience has served to enhance reliability. Predictions of line occupancy, by which selection of lines and transmission speeds were made, resulted in a communications network, which has:

- Provided response times averaging well under three seconds,
- Kept the message error rate, over conditioned lines, well within agreed limits,
- Provided as yet uninterrupted trunk line service by establishing dual lines with diverse routing,
- Allowed for future trunk line upgrading as needed, to 7200 and 9600 bps speeds.

Through the NASDAQ configuration, uninterrupted hardware performance has been ensured by the doubling of each processing unit for instant fallback support. Likewise, for the software side of NASDAQ, reliability has been designed into the functioning system itself.

Since non-technical users would be on-line to the system, entering data and updating files constantly in real-time, it was necessary to shield the processing from any disruptive entries. This is maintained by rigorous programmed control of all input. As a result, the system has not been "bombed" by any entry since regular functioning started.

Finally, the software provides extensive recovery facilities to get the system back-up and keep it running under a wide variety of adverse conditions, either temporary or of some duration. The fact that NASDAQ has been cited as "an outstanding engineering achievement" is due in large part to this reliability, now fully proven after nine months of service.

IS NASDAQ INVULNERABLE?

So it does seem as if Bunker Ramo "beat the odds" with the NASDAQ system. Despite all the things that could go wrong, everything seemed to click into place neatly. To observe the system functioning now at the Trumbull Center is to see nothing happening.

And, in that situation is hidden a new and potentially serious problem. The operators in the central site cannot avoid becoming extremely bored. There is a danger that their experience gained with a visibly struggling and trouble-loaded system in the days before start-up will now fade completely. What will happen to their ability to cope with future crises, which will, by that time, be unavoidably unique and complex? Achieving such a high level of automation can bring new problems in its wake. But since this new problem is already recognized, the remedy is on its way, we hope.

LSI Perspective—The last five years

by H. G. CRAGON

Texas Instruments
Austin, Texas

INTRODUCTION

An attempt to write the history of LSI over the past five years is a difficult task because of the lack of definition of just what is LSI—LSI means different things to different people; and there is little documented evidence on the successes and failures. I believe it is instructive to look back approximately seven years from the date of this conference to a document which, in my opinion, represents the first clear statement of the requirements for and goals of LSI.

BACKGROUND

On June 28, 1965, the Systems Engineering Group, Research and Technology Division, Air Force Systems Command issued a request for proposal for Large Scale Integrated Circuit Arrays. The problem statement and objective are quoted below.

Statement of the problem

The successful development, and subsequent production of silicon circuitry, has proven this technology capable of providing an order of magnitude improvement in reliability of systems utilizing this approach. Concurrent with this improvement in reliability has been a continuing reduction in circuit function cost. This reduction has progressed to the degree that package cost for these circuit functions represents approximately 50 percent of the final cost. It has been further shown that the intrinsic reliability of the silicon integrated circuit (SIC) is at least one more order of magnitude higher than that which is achievable with the circuits after individual packaging. This indicates that predominant failures are associated with bonding and lead failures within the circuit and multilayer wiring board or interconnection failures at the system level. These failures can be further eliminated by reduction of the circuit to circuit transfers required at the subsystem level. This in turn will greatly reduce the number of individual packages with their associated input-output terminations. Thus, it is readily apparent that if these higher order circuit functions can be achieved, today's level of reliability, cost, and size reduction can be further improved. It is the objective of this program to develop the technology for such an approach.

Objective

It is the objective of this program to accomplish as much as a ten-to-one improvement in reliability in electronic subsystem or systems over that presently available with today's integrated circuit capability. It is a further objective to achieve this improved reliability while simultaneously achieving enhanced subsystem performance and an overall cost reduction. These improvements will be accomplished through the sophisticated use of a large-scale monolithic integrated circuit array containing up to 1000 circuits utilizing either metal-oxide-semicondutor (MOS) or bipolar active devices with associated passive elements capable of being interconnected to perform logic for data processing, memory, and/or analog functions."

Several interesting points can be observed in these statements. First, the reasons for desiring LSI were to extend the reliability, reduced cost, and improved density factors that had been observed in the transition from transistor to integrated circuit equipment.

The second important point was that LSI was defined as having up to 1000 circuits using either MOS or bipolar devices for both logic, memory, and even analog functions.

Three contracts resulted in late 1965 from this Air Force initiative—one to Texas Instruments, one to RCA, and another to Philco-Microelectronics. The major thrust of the Texas Instruments effort was directed to the discretionary interconnection of the good cells on a bipolar slice for both logic and memory. RCA's program was to develop complementary MOS memory devices, while Philco-Microelectronics worked on the development of four-phase MOS shift registers.[1]

The conference proceedings of the AFIPS conferences provide a documented history of the computer industry; and, for this reason, most of my references are from this source. Many of the papers I will cite have bibliographies which will, in total, give an extensive reference to the field of LSI.

LOGIC IN LSI

The Proceedings of the Fall Joint Computer Conference of 1965 contain four papers dealing with cellular type array structures.[2,3,4,5] While these structures are interesting from an academic point of view, it is clear that the authors were concerned with devising a class of "standard" circuits which would perform the combinatorial logic function without a proliferation of part types.

Both the prospective user and the semiconductor manufacturer seemed completely preoccupied with this combinatorial logic problem during the period 1966 through 1968. L. C. Hobbs[6] stated in 1966 that "System designers must consider large-scale integrated circuits arrays as a new type of device that necessitates major revisions in systems design concepts, machine organization, and hardware software tradeoffs." At the same conference, Michael Flynn[7] made the assertion that "Integrated electronics should be much more than monolithic circuitry." Flynn believed that better solutions to the computer users problem would result if LSI were considered as a new component, and new computer systems could be devised which were based on this component.

The problems of unique parts and part types which LSI generates were studied extensively. Fubini and Smith[8] stated that "the total number of unique part numbers the industry may require for a typical group of different machine types may be as large as 100,000." They went on to suggest that the system reorganization would be useful to reduce this number of part types.

Robert Noyce[9] addressed the question of LSI cost and concluded that "Only standardization, plus the development of automatic design capabilities, will make integrated arrays available in the future at a substantial cost reduction compared with today's integrated circuits."

The recognition that an associative memory required standard logic distributed within the memory prompted several research projects in this technology. One such program was described by Ryo Igarashi and Tora Yaita[10] in 1967.

LSI MEMORY

Perhaps the seed of the first really successful LSI was planted when the computer designer's attention turned from the logic problem to the use of LSI in the memory function. Gibson's[11] paper on "*Considerations in a Block-Oriented System Design*" at the SJCC in 1967 discussed how a small but fast memory could be used to enhance the performance of a large but slow memory. In retrospect, it is obvious that this paper describes the "Cache" concept employed by IBM in some of the 360 and 370 computers. While the devices used in Mod 85 cache cannot be called LSI, a trend in the use of higher levels of integration was beginning to emerge.

At the FJCC of 1968, Conway and Spandorfer[12] discussed system enhancement applications for LSI where complete rethinking of the historical computer architecture art would not be required. Of the 10 enhancement application areas cited, 5 concerned random access memories and 1 the use of read-only memory for microprogramming.

This same conference in 1968 received a paper by Sanders[13] on semiconductor memory circuits and technology. He stated that "The manufacturing process for semiconductor memories is characterized by mass production of similar items." The industry had finally and clearly identified the first home for LSI! Since 1969, we have seen the cost per bit battle rage between the semiconductor memory manufacturer and the core memory manufacturer.

It would appear that most computer systems designers had not considered the significant cost reduction in the total cost of the hardware that would result by the use of low cost LSI memory devices. While observing that the memory represents 25 percent of a computer's main frame, we still struggled with the problems associated with replacing the 15 percent cost of the logic.[1] The core memory manufacturers were not ignoring this point, however, as core prices dropped by 30-50 percent during the 1969-1970 period.

Some of the first semiconductor memory devices were quite small; 16 bits. The technology has developed to the point where 1024-bit MOS devices and 256-bit bipolar devices are now being delivered in quantity by several manufacturers. As these devices contain the decoding logic, the problem of an excessive number of pins required for logic LSI has been circumvented.

We must conclude that these devices are true LSI as originally defined by the Air Force in 1965. We can expect that memory devices of greater capacity will continue to be introduced over the coming years.

The introduction of the IBM 370/145 with an all-semiconductor memory must establish that this technology is mature with the only question left being the upper limits on the number of bits on a chip. The memory device used is a 128-bit chip with four of these chips contained within a package giving a package of 512 bits.[15]

LSI LOGIC REVISITED

I wish to return to the question of logic in LSI. The concept of microprogramming advanced by Wilks[16] in 1951 is becoming a reality due to the availability of LSI read-only memory devices which make microprogramming an economic technique. The large number of mini-computers using microprogramming in conjunction with an ever expanding family of MSI logic devices is evidence of this success.

MSI logic devices are now being offered in excess of 100 device types by several manufacturers. These devices have provided the computer designer with the capability of designing very compact equipment using high volume production devices with the resulting low cost and high reliability. Many of size, cost, and reliability goals of the original LSI concept have been approached to a degree that seems to be acceptable from an engineer's point of view.

The quest for standard logic part types has made possible a new product that has swept the world in the last two years. This product is the electronic desk calculator. These calculators, in general, have one, two, or three MOS chips which make up the complete logic and memory function. It is difficult to trace this development in the computer literature because this product is not called a "computer." But, when one considers the complexity levels of the devices used in these calculators, one must conclude that LSI has scored another success. A typical single-chip calculator contains approximately 5000 MOS transistors on a bar having an area of approximately 25,000 square mils. This success has come about because there is a high volume requirement for a small number of part types having only a few pins. It is reported that 2,000,000 desk calculators of various types will have been produced in 1971. The fact that a device as complex as the desk calculator can be purchased for a few hundred dollars should revive the thinking on the distributed computer concept, but I will leave future projections to another author.

CONCLUSIONS

I can only conclude that LSI has been a success. When the manufacturer and the user worked together on real problems, then truly remarkable feats were accomplished. The one thousand bit memory device and the desk calculator stand as conclusive evidence. The search for applications of LSI in the logic section of the computer goes on and will perhaps find a solution which has economic justification.

REFERENCES

1 R L PETRITZ
Current status of large-scale integration technology
AFIPS Conference Vol 31 1967 Fall Joint Computer Conference pp 65-85

2 R C MINNICK
Cobweb cellular arrays
AFIPS Conference Proceedings Vol 27 Part 1 1965 Fall Joint Computer Conference pp 327-341

3 R H CANADAY
Two-dimensional interactive logic
AFIPS Conference Proceedings Vol 27 Part 1 1965 Fall Joint Computer Conference pp 343-353

4 R A SHORT
Two-rail cellular cascades
AFIPS Conference Proceedings Vol 27 Part 1 1965 Fall Joint Computer Conference pp 355-369

5 B T McKEEVER
The associative memory structure
AFIPS Conference Proceedings Vol 27 Part 1 1965 Fall Joint Computer Conference pp 371-388

6 L C HOBBS
Effects of large arrays on machine organization and hardware/software tradeoffs
AFIPS Conference Proceedings Vol 29 1966 Fall Joint Computer Conference pp 89-96

7 M J FLYNN
A prospectus on integrated electronics and computer architecture
AFIPS Conference Proceedings Vol 29 1966 Fall Joint Computer Conference pp 97-103

8 E G FUBINI M G SMITH
Limitations in solid-state technology
IEEE Spectrum May 1967 pp 55-59

9 R N NOYCE
A look at future costs of large integrated arrays
AFIPS Conference Proceedings Vol 29 1966 Fall Joint Computer Conference pp 111-114

10 R IGARASHI T YAITA
An integrated MOS transistor associative memory system with 100 ns cycle time
AFIPS Conference Proceedings Vol 30 1967 Spring Joint Computer Conference pp 499-506

11 D H GIBSON
Considerations in block-oriented systems design
AFIPS Conference Proceedings Vol 30 1967 Spring Joint Computer Conference pp 75-80

12 M E CONWAY L M SPANDORFER
A computer system designer's view of large-scale integration
AFIPS Conference Proceedings Vol 33 Part One 1968 Fall Joint Computer Conference pp 835-845

13 W B SANDER
Semiconductor memory circuits and technology
AFIPS Conference Proceedings Vol 33 Part Two 1968 Fall Joint Computer Conference pp 1205-1211

14 M G SMITH W A NOTZ
Large-scale integration from the user's point of view
AFIPS Conference Proceedings Vol 31 1967 Fall Joint Computer Conference pp 87-94

15 J K AYLING R D MOORE
Main monolithic memory
IEEE Journal of Solid-State Circuits Vol SC-6 No 5 October 1971 pp 276-279

16 M V WILKS
The best way to design an automatic calculating machine
Manchester University Computer Inaugural Conference July 1951 pp 16-18

Toward more efficient computer organizations*

by MICHAEL J. FLYNN

The Johns Hopkins University
Baltimore, Maryland

Two significant trends have evolved which will substantially effect the nature of computer organizations over the next ten years:

1. The almost universal use of higher level programming languages and corresponding decrease in the use of machine level programming.
2. The continued improvement in cost and performance of both logic and memory technologies.

These trends portend many changes. Most of them gradual, but certainly effecting the nature and perspective that we currently hold of computer organizations and their future evolution.

The machine language programmer of today does not program in assembly code for convenience. He codes almost exclusively for reasons of efficiency requirements, that is, the control of the physical resources in an optimal way. Ease of use is not as important a consideration in the development of a machine language as efficiency. This coupled with technological improvements, especially in the area of memory, portend the first significant change in computer organizations: the increase in use of parallel "or concurrent" operation of the resources of the systems. Since the hardware is inexpensive, units are made autonomous in the hope that their concurrent operation will produce more efficient program performances. "Efficient" in this sense is the respect to the user program and its performance, not with respect to the busyness of some internal subcomponent of the system.

There are several forms of parallelism: First, one may have concurrency of operation within a (internal) Single Instruction stream-Single Data stream (conventional type) program. In fact, this concurrency may or may not be transparent to the programmer. The second form of parallelism involves the use of multiple streams in programs.

IMPLICIT VS EXPLICIT PARALLELISM

Implicit concurrency or parallelism may be grossly characterized as that degree of concurrency, achieved internal to the processors, which is not visible to its programmer. Thus explicit parallelism is that concurrency of functional execution which is visible to the programmer'and demands his attention and exploitation. To date most attempts to take advantage of the two earlier mentioned trends have used a form of implicit parallelism. The machine designers attempted to achieve a certain concurrency of execution which is transparent to the problem program (machine language) by "wiring in" the concurrency control and preserving the use of older machine languages; thus the existence of the cache type[1] memory, as well as heavily overlapped computer organizations.[2]

Cache memory involves the use of high speed buffer memory to contain recently used blocks of main memory storage in an attempt to predict the subsequent memory usage requirements—thus saving the slow access time to storage.

The overlap organization involves the multiple issuing of instructions concurrently. When more than one instruction is in process at a given time, care must be taken that the source and sink data are properly interlocked.

Notice that much of the implicit parallelism is a direct result of the present nature of the machine language instruction. Rather than change the definition of the present machine language instruction to make it more complex, the manufacturers essentially standardize on it and use it as a universal intermediate programming language. Thus it has the aspect of transportability not usually found in other languages. Notice the ma-

* This work was supported, in part, by the U. S. Atomic Energy Commission under contract AT(11-1)-3288.

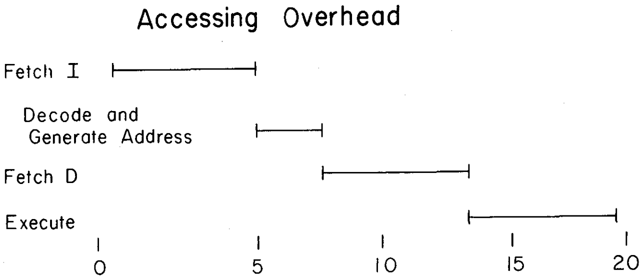


Figure 1—Conventional instruction

chine language instruction is not an ideal vehicle to allow one to control the physical resources of the system since the concurrent control is achieved automatically, without the possibility of any global optimization.

EXPLICIT PARALLELISM*

How can the control of the resources of the system be made visible to the programmer? First let us review the evolution of the machine language as we know it and compare it with microprogrammed systems.

An essential feature of a microprogram system is the availability of a fast storage medium. "Fast" in this sense is used with respect to ordinary logical combinational operations. That is, the access time of the storage device is of the same order as the primitive combinational operations that the system can perform—the cycle time of the processor. Another important attribute of modern microprogrammed systems is that this "fast" storage is also writable, establishing the concept of "dynamic microprogramming." It is the latter that distinguishes the current interest in microprogrammed systems from earlier attempts using READ-ONLY memory.[3–5] Consider the timing chart for a conventional (nonmicroprogram) machine instruction (Figure 1). Due to slow access of instruction and data, a substantial amount of instruction execution time is spent in these overhead operations. Contrast this with the situation shown in Figure 2, illustrating the microprogram's instruction. In Figure 2 there is an implicit assumption of homogeneity of memory, that is, that data and operands are all located in the same fast storage as the instructions. The latter assumption is valid when the storage is writable.

The preceding implies another significant difference in conventional machines and microprogrammed machines—the power of the operation to be performed by the instruction. In the conventional machine, given the substantial accessing overhead, it is very important to orient the operation so that it will be as significant as possible in computing a result. To do otherwise would necessarily involve a number of accessing overheads; thus we have the evolution of the rich and powerful instruction sets of the second and third generation machines. With dynamically microprogrammed systems (Figure 2), the powerful operation will do no more than a sequence of steps. Indeed, the rich instruction operation may (if its richness is done at the expense of flexibility) cause an overall degradation in system performance.

EXPLICIT INTERNAL PARALLELISM (see Figure 3)

In the traditional system, since the overhead penalties are so significant, there is little real advantage in keeping the combinational resources of the system simultaneously busy or active. This would mean that perhaps 17 or 18 cycles would be required to perform an instruction rather than 20—hardly worth the effort. A much different situation arises with the advent of dynamic microprogramming.[6] Consider a system partitioned into resources, e.g., adder, storage resources, addressing resources, etc. If the microinstruction can be designed in such a fashion that during a single execution one can control the flow of data internal to each one of these resources, as well as communicate between them, significant performance advantages can be derived. Of course these advantages come at the expense of a wider microinstruction. The term horizontal microinstruction has been coined to describe the control for this type of logical arrangement.

RESIDUAL-CONTROL IN DYNAMIC MICROPROGRAMMING

The preceding scheme sacrificed storage efficiency in order to achieve performance. Schemes which attempt

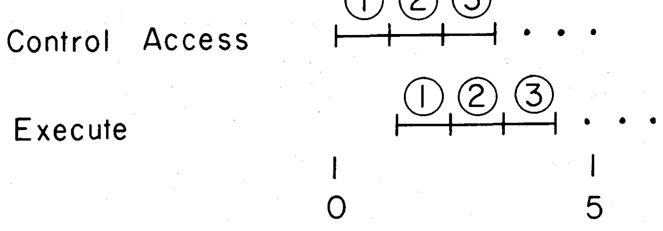


Figure 2—Microprogrammed execution

[*] The remarks on microprogramming are abstracted from Reference 15.

to achieve efficiencies both in storage and performance can be conceived, based upon an information theoretic view of control and storage. Much information specified in the microinstruction is static, i.e., it represents environmental information. The status remains unchanged during the execution of a number of microinstructions. Many of the gating paths and adder configurations remain static for long periods of time during the emulation of a single virtual machine. If this static information and specification is filtered out of the microinstruction and placed in "setup" registers, the combination of a particular field of a microinstruction with its corresponding setup register, would completely define the control for resource (Figure 4). This sacrifices nothing in flexibility—it allows for a more efficient use of storage.

EXTERNAL PARALLELISM—SIMD

Explicit parallelism may be internal as in the earlier discussion or external, that is, where the programmer directly controls a number of different items of data (Data Streams) or has a number of programs executed simultaneously (Instruction Streams). We refer to the first type of organization as a single instruction stream, multiple data stream (SIMD).[2] There are three kinds of SIMD machine organizations currently under development: (1) the array processor,[7] (2) the pipeline processor,[8] (3) the associative processor.[9]

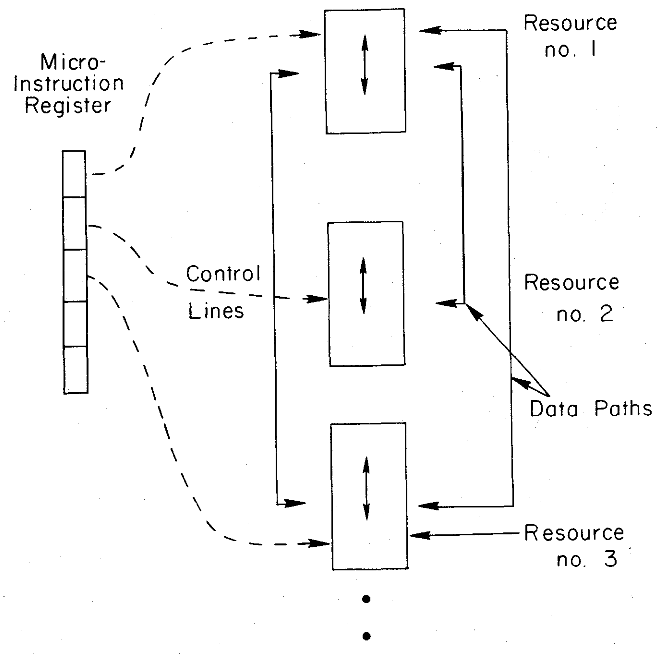


Figure 3—Simultaneous resource management

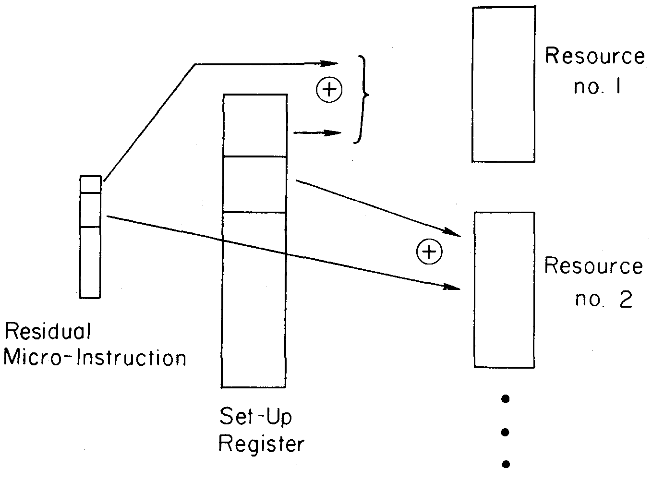


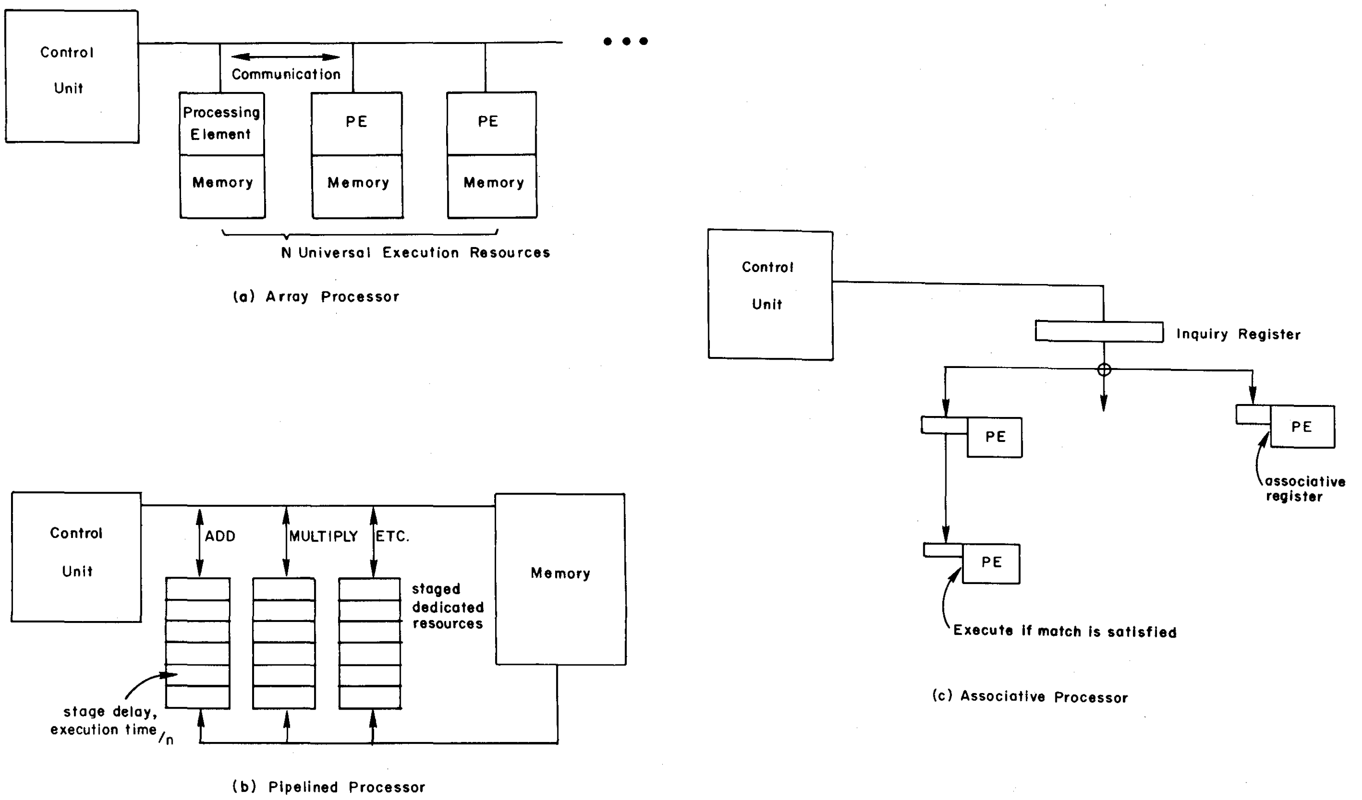
Figure 4—Residual control in microprogramming

While these systems differ basically in their organization of resources and their communication structures, their gross programming characterizations are quite similar. One instruction is issued for essential concurrent execution on a number of operand data pairs. Figure 5 illustrates each of the three organizations. In the array processor the execution resources are replicated n times. Since each set of resources is capable of executing an operand pair, in a sense each resource set is an independent slave processor. It may indeed (and usually does) have its own private memory for holding operand pairs. Since each processing element is physically separate, communications between the elements can be a problem. At least it represents an overhead consideration.

The pipelined processor is in many ways similar to the array processor, except that the execution resources are pipelined. That is, they are staged in such a way that the individual staging time for a pair of operands is $1/n$ of the control unit processing time. Thus n pairs of operands are processed in one control unit time. This has the same gross structure (from a programming point of view) as an array processor. But since the pairs of operands reside in a common storage (since the execution units are common) the communications overhead occur in a different sense. It is due to required rearranging of data vectors in proper order so that they may be scanned out at a rate compatible to the pipelined rate of the execution resources.

Both array processors and pipelined processors work over directly addressed data pairs. Thus, the location of each pair of operands is known in advance before being processed.

The associative processor is similar in many ways to the array processor except that the execution of a



(a) Array Processor

(b) Pipelined Processor

(c) Associative Processor

Figure 5—Some idealized SIMD processor (a, b, & c)

control unit instruction is conditional upon the slave unit matching an inquiry pattern that is presented to all units. Matching here should be interpreted in a general sense, that is, satisfying some specified condition with respect to an inquiry pattern. Each slave unit contains at least one pattern register which is compared to the inquiry prior to conditional execution.

Each of these processor types performs well within a certain class of problem. The dilemma has been to extend this class by restructing algorithms. It has been suggested (by M. Minsky,[17] B. Arden, et al.) that instead of an ideal of n fold performance (on data streams) improvement over a sequential processor, the SIMD processor is more likely to realize

$$\text{Perf} \rightarrow \log_2 n$$

A possible rational for this is discussed elsewhere.[16]

EXTERNAL PARALLELISM—MIMD

The multiple instruction-stream multiple data-stream programs are merely statements of independency of tasks—for use on "multiprocessors." I believe that exciting new developments are occurring in the multiprocessor organization through the use of shared resources.

Two traditional dilemmas for a multiprocessor (MIMD) computer organization are: (1) that of improving performance at a rate greater than linear with respect to the increasing number of independent processors involved, and (2) providing a method of dynamic reconfiguration and assignment of the instruction handling resources to match changing program environments. Shared Resource Organizational arrangements may allow solution of these problems.

SHARED EXECUTION RESOURCES

The execution resources of a computer (adders, multiplier, etc.—most of the system exclusive of registers and minimal control) are rarely efficiently used. This is especially true in modern large systems such as CDC7600 and IBM Model 195. Each execution facility is capable of essentially processing, at the maximum rate, an entire program consisting of instructions which use that resource. The reason for this design is that, given that instruction I_{i-1} required a floating point add type resource, it is probably that I_i will also—rather than, say a Boolean resource. Indeed the "execution bandwidth" available may exceed the actual instruction rate by a factor of 10.

Another factor aggravating the above imbalance is the development of arithmetic pipelining internal to the resources. In these arrangements, the execution function itself is pipelined, as per the pipeline processor. This provides a more efficient mechanism than replication for attaining a high execution rate.

SKELETON INSTRUCTION UNITS

In order to effectively use this execution potential, we postulate the use of multiple skeleton processors sharing the execution resources. A "skeleton" processor consists of only the basic registers of a system and a minimum of control (enough to execute a load or store type instruction). From a program point of view, however, the "skeleton" processor is a complete and independent logical computer.

These processors may share the resources in space,[10] or time.[11, 12, 13] If we completely rely on space sharing, we have a "cross-bar" type switch of processors—each trying to access one of n resources. This is usually unsatisfactory since the "cross-bar" switching time overhead can be formidable. On the other hand, time phased sharing (or time multiplexing) of the resources can be attractive, in that no switching overhead is involved and control is quite simple if the number of multiplexed processors are suitably related to each of the pipelining factors. The limitation here is that again only one of the m available resources is used at any one moment.

We suggest that the optimum arrangement is a combination of space-time switching (Figure 6). The time factor is the number of "skeleton" processors multiplexed on a time phase ring while the space factor is the number of multiplexed processor "rings," K, which simultaneously request resources. Note that K processors will contend for the resources and, up to K-1, may be denied serivce at that moment. Thus rotating priority among the rings is suggested to guarantee a minimum performance. The partitioning of the resources will be determined by the expected request statistics.

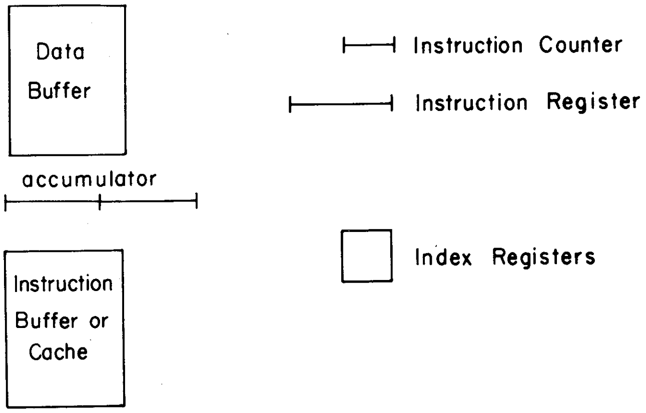


Figure 6a—Skeleton processor

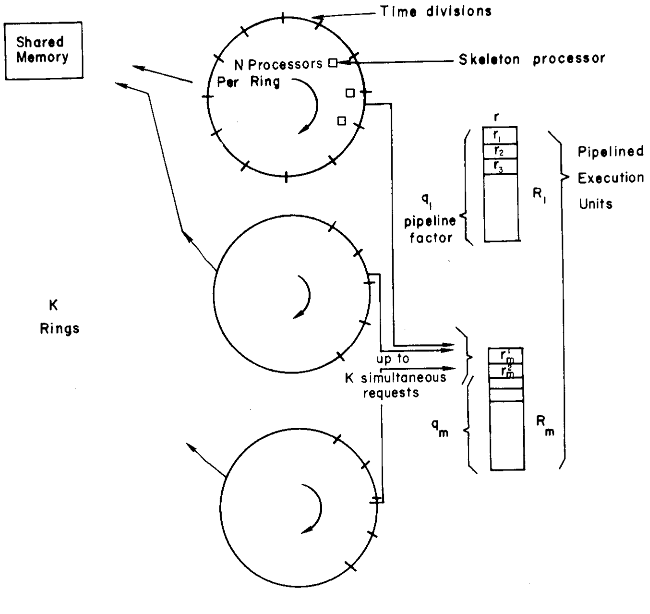


Figure 6b—Time-multiplexed multiprocessing

SUB-COMMUTATION

When the amount of "parallelism" (or number of identifiable tasks) is less than the available processors, we are faced with the problem of speeding up these tasks. This can be accomplished by designing certain of the processors in each ring with additional staging and interlock[14] (the ability to issue multiple instructions simultaneously) facilities. The processor could issue multiple instruction execution requests in a single ring revolution. For example, in a ring of $N = 16$

—8 processors could issue two request/revolutions or
—4 processors could issue four request/revolutions or
—2 processors could issue eight request/revolutions or
—1 processor could issue sixteen request/revolutions.

This partition is illustrated in Figure 7.

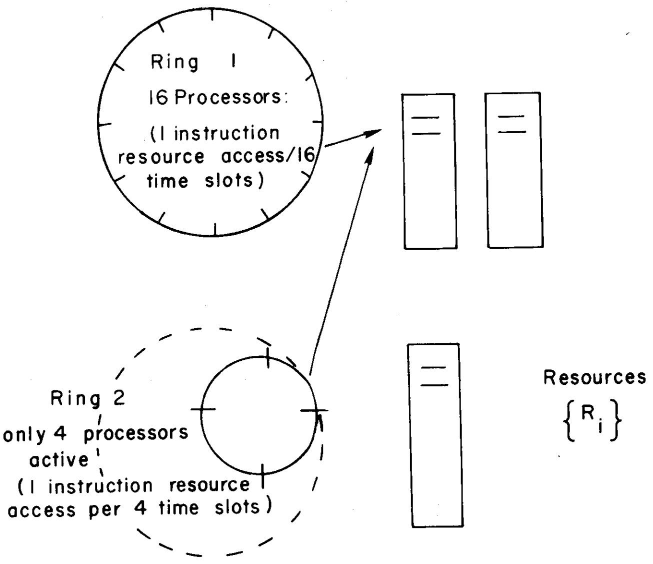

Ring 1
16 Processors:
(1 instruction resource access/16 time slots)
Ring 2
only 4 processors active
(1 instruction resource access per 4 time slots)
Resources
$\{R_i\}$

Figure 7—Sub-commutation of processors on a ring

IMPLEMENTATION AND SIMULATION

A proposed implementation of this shared resource multiprocessor is described elsewhere, together with a detailed instruction by instruction simulation.[13]

The system consisted of eight skeleton processors per ring and four rings per system. The execution resources were generally pipelined by eight stages. The proposed system had a maximum capability of over about 500 million instructions per second and was capable of operating at over eighty percent efficiency (i.e., performance was degraded by no more than 20 percent from the maximum).

THE TECHNOLOGY OF OPERATING SYSTEMS

Yet another type of parallelism involves the concurrent management of the gross (I/O, etc.) resources of the system by the operating system.

The evolution of very sophisticated operating systems and their continued existence is predicated on one essential technological parameter: the latency of access of data from an electro-mechanical medium. Few other single parameters in computing systems have so altered the entire face and approach of computing systems. Over the period of time, 1955 to 1971, access time to a rotating medium has remained essentially constant between 20 to 100 m seconds. Processing speeds, on the other hand, have increased by almost a factor of 10,000. Thus, originally we had a very simple system of scheduling. Data sets and partitions of programs were processed by simply waiting in turn as each request was processed. Today's situation requires scheduling essentially 1,000 times more sophisticated. In other words, if more than one instruction in 1,000 requires a machine to go to an idle state, we would have the same system performance, as we had in 1955. Thus the nature of the dilemma. If a technological solution, on the other hand, could be achieved with low latency for accessing large data sets, the whole need for sophisticated operating systems to sustain job queues and I/O requests would be diminished if not eliminated altogether. A great deal of time and money has been spent in analysis and synthesis of complex resource scheduling, resource control operating systems. I surmise that if a small fraction of these resources had been directly addressed to the physical problem of latency, physical solutions would have been found which would have resulted in much more efficient systems.

This remains one of the great open challenges of the next five years.

CONCLUSIONS

With the adoption of higher level languages for user convenience and the evolution of machine language for efficient control of resources, a departure in user oriented machine level instructions is seen. This coupled with the increasing performance of memory portends a major shift in the structure of the conventional machine instruction. A new structure will allow the program explicit control over the resources of the system.

With the increasing improvement in logic technology cost-performance, more specialized and parallel organizations will appear. For specialized applications SIMD type organization in several variations will appear for dedicated applications. The more general organization MIMD (particularly of shared resource type) appeared even more interesting for general purpose applications.

The major technological problems for the next five years remain in the memory hierarchy. The operating systems problems over the past years are merely a reflection of this. A genuine technological breakthrough is needed for improved latency to mass storage devices.

REFERENCES

1 D GIBSON
Considerations in block-oriented system design
AFIPS Conf Proceedings Vol 13 pp 75-80

2 M FLYNN
Very high-speed computing systems
Proceedings of the IEEE Dec 1966 p 1901

3 M V WILKES
The growth of interest in microprogramming—A literature survey
Computing Surveys Vol 1 and 3 Sept 1969 pp 139-145

4 M J FLYNN M D MacLAREN
Microprogramming revisited
Proceedings Assoc Comput Mach National Conference 1967 pp 457-464

5 R F ROSIN
Contemporary concepts in microprogramming and emulation
Computing Surveys Vol 1 and 4 Dec 1969 pp 197-212

6 R W COOK M J FLYNN
System design of a dynamic microprocessor
IEEE Transactions on Computers Vol C-19 March 1970 pp 213-222

7 G H BARNES et al
The Illiac IV computer
IEEE Transactions on Computers Vol C-17 No 8 August 1968 pp 746-757

8 ______
CDC STAR-100 reference manual

9 J A GITHENS
An associative, highly-parallel computer for radar data processing
Parallel Processor Systems L C Hobbs et al Editors Pub Spartan Press NY 1970 pp 71-86

10 P DREYFUS
Programming design features of the GAMMA 60 computer
Proc EJCC Dec 1958 pp 174-181

11 J E THORTON
Parallel Operation in Control Data 6600
AFIPS Conference Proceedings Vol 26 Part II FJCC 1964 pp 33-41

12 R A ASHENBRENNER M J FLYNN G A ROBINSON
Intrinsic multiprocessing
AFIPS Conference Proceedings Vol 30 SJCC 1967 pp 81-86

13 M J FLYNN A PODVIN K SHIMIZU
A multiple instruction stream processor with shared resources
Proceedings of Conference on Parallel Processing Monterey California 1968 Pub Spartan Press 1970

14 G S TJADEN M J FLYNN
Detection and execution of independent instructions
IEEE Transactions on Computers October 1970 Volume C-19 No 10 pp 889-895

15 M J FLYNN R ROSIN
Microprogramming: An introduction and a viewpoint
IEEE Transactions on Computers July 1971 Vol C-20 No 7 pp 727-731

16 M J FLYNN
On computer organizations and their effectiveness
IEEE Transactions on Computers To be published

17 M MINSKY S PAPERT
On some associative, parallel, and analog computations
Associative Information Techniques Ed by E J Jacks Pub American Elsevier 1971

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

AFIPS OFFICERS AND BOARD OF DIRECTORS

President

Mr. Keith Uncapher
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Vice President

Mr. Walter L. Anderson
General Kinetics, Inc.
11425 Isaac Newton Square, South
Reston, Virginia 22070

Secretary

Dr. Donald Walker
Artificial Intelligence Group
Stanford Research Institute
Menlo Park, California 94025

Treasurer

Dr. Robert W. Rector
University of California
6115 Mathematical Sciences Building
Los Angeles, California 90024

Executive Director

Dr. Bruce Gilchrist
AFIPS
210 Summit Avenue
Montvale, New Jersey 07645

ACM Directors

Mr. Walter Carlson
IBM Corporation
Armonk, New York 10504

Mr. Donn B. Parker
Stanford Research Institute
Menlo Park, California 94025

Dr. Ward Sangren
The University of California
521 University Hall
2200 University Avenue
Berkeley, California 94720

IEEE Directors

Dr. A. S. Hoagland
IBM Corporation
P.O. Box 1900
Boulder, Colorado 80302

Dr. Robert A. Kudlich
Raytheon Co., Equipment Division
Wayland Laboratory
Boston Post Road
Wayland, Massachusetts 01778

Professor Edward J. McCluskey
Stanford University
Department of Electrical Engineering
Palo Alto, California 94305

Simulations Council Director

Mr. Frank C. Rieman
Electronic Associates, Inc.
P.O. Box 7242
Hampton, Virginia 23366

Association for Computation Linguistics Director

Dr. A. Hood Roberts
Center for Applied Linguistics
1717 Massachusetts Avenue, N.W.
Washington, D.C. 20036

American Institute of Aeronautics and Astronautics Director

Mr. Frank Riley, Jr.
Auerbach Corporation
1501 Wilson Boulevard
Arlington, Virginia 22209

American Institute of Certified Public Accountants Director

Mr. Noel Zakin
Computer Technical Services
ACIPA—666 Fifth Avenue
New York, New York 10019

American Statistical Association Director

Dr. Mervin E. Muller
5303 Mohican Road
Mohican Hills
Washington D.C. 20016

Instrument Society of America Director

Mr. Theodore J. Williams
Purdue Laboratory for Applied Industrial Control
Purdue University
Lafayette, Indiana 47907

Society for Information Display Director

Mr. William Bethke
RADC (EME, W. Bethke)
Griffis Air Force Base
Rome, New York 13440

American Society for Information Science Director

Mr. Herbert Koller
ASIS
1140 Connecticut Avenue, N.W. Suite 804
Washington, D.C. 20036

Society for Industrial and Applied Mathematics Director

Dr. D. L. Thomsen, Jr.
IBM Corporation
Armonk, New York 10504

Special Libraries Association Director

Mr. Burton E. Lamkin
Office of Education—Room 5901
7th and D Streets, S.W.
Washington, D.C. 20202

JOINT COMPUTER CONFERENCE BOARD

President

Mr. Keith W. Uncapher
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Vice President
Mr. Walter L. Anderson
General Kinetics, Incorporated
11425 Isaac Newton Square, South
Reston, Virginia 22070

Treasurer
Dr. Robert W. Rector
University of California
6115 Mathematical Sciences Building
Los Angeles, California 90024

ACM Representative

Mr. Richard B. Blue Sr.
1320 Victoria Avenue
Los Angeles, California 90019

IEEE Representative
Mr. Stephen S. Yau
Northwestern University
Evanston, Illinois 60201

SCI Representative
Mr. Paul Berthiaune
Electronic Associates, Inc.
West Long Branch, New Jersey 07764

JOINT COMPUTER CONFERENCE COMMITTEE

Mr. Jerry L. Koory, Chairman
H-W Systems
525 South Virgil
Los Angeles, California 90005

JOINT COMPUTER CONFERENCE TECHNICAL PROGRAM COMMITTEE

Mr. David R. Brown, Chairman
Stanford Research Institute
333 Ravenswood Avenue
Menlo Park, California 94025

FUTURE JCC CONFERENCE CHAIRMAN

Dr. Robert Spinrad
Xerox Data Systems
701 South Aviation Blvd.
El Segundo, California 90245

1972 SJCC STEERING COMMITTEE

General Chairman

John E. Bertram
IBM Corporation

Vice Chairman

Emanuel S. Savas
The City of New York

Secretary

Carole A. Dmytryshak
Bankers Trust Company

Controller

F. G. Awalt, Jr.
IBM Corporation

Technical Program

Jacob T. Schwartz—Chairman
Courant Institute
New York University
Kenneth M. King—Vice Chairman
City University of New York

Member

Donald L. Thomsen, Jr.
IBM Corporation

Local Arrangements

Dorothy I. Tucker—Chairman
Bankers Trust Company
Seymour Weissberg—Vice Chairman
Bankers Trust Company

Registration

Jerome Fox—Chairman
Polytechnic Institute of Brooklyn
Sol Sherr—Vice Chairman
North Hills Associates

Conference Publications

Charles G. Van Vort—Chairman
Society of Automotive Engineers
Allan Marshall—Vice Chairman
Society of Automotive Engineers

Exhibits

J. Burt Totaro—Chairman
Datapro Research
John L. Sullivan—Vice Chairman
Hewlett-Packard Company

Special Activities

Robert J. Edelman—Chairman
Western Electric Company
R. C. Spieker—Vice Chairman
A.T.&T. Company

Public Relations

Walter C. Fleck
IBM Corporation

ACM Representative

Paul M. Wodiska
Hoffman La Roche

IEEE Computer Society Representative

Stephen Pardee
Bell Telephone Laboratories

SCI Representative

Arthur Rubin
Electronic Associates, Inc.

JCC Committee Liaison

Harry L. Cooke
RCA Laboratories

SESSION CHAIRMAN, REVIEWERS AND PANELISTS

SESSION CHAIRMEN

Aho, Alfred
Belady, Laszlo A.
Bell, James R.
Carroll, Donald C.
Chang, Sheldon S. L.
Conte, Samuel D.
Crocker, Stephen
Denning, Peter
Dolotta, T. A.
Flores, Ivan
Flynn, Michael J.
Fosdick, Lloyd D.
Fuchs, Edward
Gale, Ronald M.
Galler, Bernard
Genik, Richard J.
Graham, Robert
Gross, William A.
Hamblen, John
Karp, Richard
Kurtz, Thomas
Lehmann, John R.
Lewin, Morton H.
Merwin, Richard
Morris, Michael
Newton, Carol M.
Papert, Seymour
Pennington, Ralph
Potts, Jackie S.
Radke, Charles
Ramsay, W. Bruce
Rosen, Saul
Sammet, Jean
Schwartz, Jules I.
Sonquist, John
Stenberg, Warren
Sturman, Gerald
Traub, Joseph F.
Zimbel, Norman S.

REVIEWERS

Ackerman, Eugene
Agnew, Palmer
Anderson, Sherwood
Atkinson, Richard C.
Baldwin, John
Barry, Patrick
Bartlett, W. S.
Bayles, Richard
Bellucci, Jim
Bernstein, Morton I.
Bork, Alfred
Brawn, Barbara S.
Bredt, Thomas H.
Brender, Ronald
Bryan, G. Edward
Cannizzaro, Charles
Carter, William
Cederquist, Gerald N.
Chang, H.
Chapman, Barry L.
Cheng, David D.
Chu, W. W.
Dorato, Peter
Earley, Jay
Elashoff, Robert
Enger, Norman
Ferrari, Domenico
Fischer, M. J.
Follett, Thomas
Fox, Margaret R.
Fraser, A. G.
Freiman, Charles
Frye, Charles
Fuller, Samuel H.
Garland, Stephen J.
Gebert, Gordon
Gee, Helen
Gelenbe, S. E.
Gorenstein, Samuel
Graham, G. Scott
Graham, Susan
Gray, James N.
Griffith, John
Halpern, Mark
Halstead, Maurice
Hamilton, James A.
Harker, J.
Harrison, M. A.
Hebalkar, Prakash
Hennie, F. C.
Hight, S. L.
Hildebrand, David B.
Hochdorf, Martin
Hoffman, David
Hsiao, Mu Y.
Huseth, Richard
Jessep, Donald C.
Kaman, Charles
Karp, Peggy
Katzenelson, Jacob
Keller, Robert M.
Kirn, Fred L.
Kohlhaas, Charles A.
Kosinski, Paul
Laurance, Neal L.
Lavenberg, Stephen S.
Lechner, Bernard
Lehman, Meir M.
Levy, S. Y.
Lundquist, Alfred
MacDougall, M. H.
Marks, Gregory A.
Martin, R. L.
Mayham, Albert
McClung, Charles R.
Meissner, Loren
Miller, Edward F., Jr.
Moore, Scott E.
Morris, James
Mulford, James
Ninke, William
Oehler, Richmond
Oliver, Paul
Paige, M. R.
Peng, Theodore
Rabinowitz, I. N.
Reddy, Raj
Richman, Paul
Roberts, John D.
Robertson, David
Rosenkrantz, D. J.
Rosenthal, Neal
Salbu, E.
Salerno, John
Salz, J. S.
Sattley, Kirk
Scarborough, Collin W.
Schneider, Victor B.
Schroeder, Michael
Seader, David
Shore, J.
Siler, William
Silverman, Harvey S.
Slamecka, Vladimir

Slutz, Donald R.
Smith, O. Dale
Smith, Robert B.
Smith, Stephen E.
Smith, W.
Spirn, Jeffrey R.
Steiglitz, Ken
Stone, Harold S.
Stonebraker, Michael R.
Stubblefield, C. W.
Thatcher, J. W.
Thornton, James
Traiger, Irving L.
Tung, Frank C.
Ullman, J. D.
Urion, Henry K.
Van Norton, Roger
Verhotz, Robert
Wadia, A. B.
Walsh, Donald J.
Ward, John
Wildmann, M.
Williams, John
Williams, Theodore J.
Woo, Lynn S.
Wu, Y. S.
Young, Paul

PANELISTS

Avizienis, Algirdas
Bell, C. Gordon
David, Edward, Jr.
Fernback, Sidney
Fox, Margaret R.
Hutt, Arthur E.
Kohl, Herbert
Levien, Roger
McCluskey, Edward J.
Minsky, Marvin
Pratt, Arnold W.
Rosenthal, Neal
Rusk, S. K.
Savides, Peter
Slamecka, Vladimir
Suppes, Patrick
Wilcox, Lawrence

PRELIMINARY LIST OF EXHIBITORS

Addison-Wesley Publishing Company, Inc.
Addressograph Multigraph Corporation
AFIPS Press
American Elsevier Publishing Co., Inc.
Auerbach Corporation
Auricord Div.—Conrac Corporation
Badger Meter, Inc., Electronics Div.
Bell & Howell, Electronics and Instruments Group
Benwill Publishing Corporation
Burroughs Corporation
Cambridge Memories, Inc.
Centronics Data Computer Corporation
Codex, Inc.
ComData Corporation
Computer Copies Corporation
Computer Decisions and Electronic Design Magazines
Computer Design Corporation
Computer Design Publishing Corporation
Computer Investors Group, Inc.
Computer Operations, Inc.
Computer Transceiver Systems, Inc.
Creative Logic Corporation
Data Disc, Inc.
Data General Corporation
Data Interface Associates
Datamation
Data Printer Corporation
Datapro Research Corporation
Decision Data Computer Corporation
Diablo Systems, Inc.
Digi-Data Corporation
Digital Computer Controls
Digitronics Corporation
DuPont Company
Eastern Specialites Co., Inc.
Electronic Associates, Inc.
Electronic News—Fairchild Publications
Extel Corporation
Fabri-Tek Incorporated
Facit-Odhner, Inc.
General Radio Company
Gould, Inc., Instrument Systems Div.
GTE Lenkurt
G-V Controls, Div. of Sola Basic Ind.
Hewlett Packard Company
Hitchcock Publishing Company
Houston Instrument
IMSL
Incoterm Corporation
Inforex, Inc.
International Teleprinter Corp.
Iron Mountain Security Storage Corp.
Kennedy Company
KYBE Corporation
Litton ABS OEM Products
Lockheed Electronics Co., Inc.
Lorain Products Corporation
3M Company
Micro Switch, A Div. of Honeywell
Milgo Electronic Corporation
Miratel Division—BBRC
MITE Corporation
Modern Data Services, Inc.
McGraw-Hill
NCR Special Products Division
Nortronics Company, Inc.
Numeridex Tape Systems
ODEC Computer Systems, Inc.
Optical Scanning Corporation
Panasonic
Paradyne Corporation
Penril Data Communications, Inc.
Peripherals General, Inc.
Pioneer Electronics Corporation
Prentice Hall, Inc.
Princeton Electronic Products, Inc.
Printer Technology, Inc.
Quindata
Raymond Engineering Inc.
Raytheon Company
RCA Corporation
Remex—A unit of Ex-Cell-O Corp.
RFL Industries, Inc.
Sanders Data Systems, Inc.
Sangamo Electric Company
Science Accessories Corporation
The Singer Company
Sola Electric Division
Sonex, Inc.—I/Onex Div.
Sorbus, Inc.
Spartan Books
Standard Memories, Inc.
Storage Technology Corporation
Sugarman Labs.
The Superior Electric Company
Sykes Datatronics, Inc.
Tandberg of America, Inc.
Techtran Industries, Inc.
Tektronix, Inc.
Tele-Dynamics—Div. of AMBAC Inds., Inc.
Teletype Corporation
Texas Instruments Incorporated
Unicomp, Inc.
United Air Lines
United Telecontrol Electronics, Inc.
Van San Corporation
Vogue Instrument Corporation
Webster Computer Corporation
Wells T.P. Sciences, Inc.
John Wiley & Sons, Inc.
XLO Computer Products

AUTHOR INDEX