

AFIPS

CONFERENCE
PROCEEDINGS


VOLUME 29

1966

FALL JOINT
COMPUTER
CONFERENCE

NOVEMBER 7-10
SAN FRANCISCO, CALIFORNIA

1966
SPARTAN BOOKS
Washington, D. C.



The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1966 Fall Joint Computer Conference Committee or the American Federation of Information Processing Societies.

Library of Congress Catalog Card Number 55-44701
Spartan Books, Div. of
Books, Inc.
1250 Connecticut Avenue N.W.
Washington, D. C.

© 1966 by the American Federation of Information Processing Societies, 211 E. 43rd St., New York, N. Y. 10017. All rights reserved. This book, or parts thereof, may not be reproduced in any form without permission of the publishers.

Sole distributors in Great Britain, the British
Commonwealth, and the Continent of Europe:

Macmillan Co., Ltd.
4 Little Essex Street
London W.C. 2

CONTENTS

TIME-SHARING PROCESSORS AND EXECUTIVE SYSTEMS

A Conversational System for Incremental Compilation and Execution in a Time-Sharing Environment	J. L. RYAN R. L. CRANDALL M. MEDWEDEFF	1
Performance of a Monitor for a Real-Time Control System	E. S. HOOVER B. J. ECKHART	23
On-Line Debugging Techniques: A Survey	T. G. EVANS D. L. DARLEY	37
The SDS SIGMA 7: A Real-Time, Time-Sharing Computer	M. J. MENDELSON A. W. ENGLAND	51

INTEGRATED ELECTRONICS AND THE FUTURE OF COMPUTERS

Technological Foundations and Future Directions of Large-Scale Integrated Electronics	R. L. PETRITZ	65
Effects of Large Arrays on Machine Organization and Hardware/Software Tradeoffs	L. C. HOBBS	89
A Prospectus on Integrated Electronics and Computer Architecture	M. J. FLYNN	97
The System/Semiconductor Interface with Complex Integrated Circuits	W. B. SANDER	105
A Look at Future Costs of Large Integrated Arrays	R. NOYCE	111

COMPUTERS AND PUBLISHING

A Multiprogrammed Teleprocessing System for Computer Typesetting	B. E. NEBEL	115
Integrated Automation in Newspaper and Book Production	J. H. PERRY, JR.	125
A Special Purpose Computer for High-Speed Page Composition	C. J. MAKRIS	137
Computerized Typesetting of Complex Scientific Material	J. H. KUNEY B. G. LAZORCHAK S. W. WALCAVICH D. SHERMAN	149
A Computer-Assisted Page Composing System	G. Z. KUNKEL	157

HYBRID COMPUTERS AND RANDOM VARIABLES

A General Method for Producing Random Variables in a Computer	G. MARSAGLIA	169
A Unified Approach to Deterministic and Random Errors in Hybrid Loops	J. J. VIDAL	175
Hybrid Computer Solutions of Partial Differential Equations by Monte Carlo Methods	W. D. LITTLE	181
Parameter Optimization by Random Search Using Hybrid Computer Techniques	G. A. BEKEY M. H. CRAN A. E. SABROFF A. WONG	191

ENGINEERING DESIGN BY MAN/COMPUTER INTERACTION

A Parametric Graphical Display Technique for On-Line Use	M. I. DERTOUZOS H. L. GRAHAM	201
A System for Time-Sharing Graphic Consoles	J. R. KENNEDY	211
The Lincoln Wand	L. G. ROBERTS	223
Using a Graphic Data-Processing System to Design Artwork for Manufacturing Hybrid Integrated Circuits	J. S. KOFORD P. R. STRICKLAND G. A. SPORZYNSKI E. H. HUBACHER	229
Automated Logic Design Techniques Applicable to Integrated Circuit Technology	R. WAXMAN M. T. MCMAHON B. J. CRAWFORD A. B. DEANDRADE	247

COMPUTER MEMORIES

Cost Performance Analysis of Integrated-Circuit Core Memories	D. W. MOORE	267
A 200-Nanosecond Thin Film Main Memory System	S. A. MEDDAUGH K. L. PEARSON	281
A Rotationally Switched Rod Memory with a 100-Nanosecond Cycle Time	B. A. KAUFMAN P. B. ELLINGER H. J. KUNO	293
A 500-Nanosecond Main Computer Memory Utilizing Plated-Wire Elements	J. P. MCCALLISTER C. F. CHONG	305
A High-Speed Integrated Circuit Scratchpad Memory	I. CATT E. C. GARTH D. E. MURRAY	315
Sonic Film Memory	H. WEINSTEIN L. ONYSHKEVYCH K. KARSTAD R. SHAHBENDER	333

NATURAL LANGUAGE

English for the Computer	F. B. THOMPSON	349
An Approach Toward Answering English Questions from Text	R. F. SIMMONS J. F. BURGER R. E. LONG	357
DEACON: <i>Direct English Access and CONTROL</i>	J. A. CRAIG S. C. BEREZNER H. C. CARNEY C. R. LONGYEAR	365
Computer Assisted Interrogation	C. T. MEADOW D. W. WAUGH	381

SOME COMMUNICATIONS ASPECTS OF TIME-SHARING SYSTEMS

Some Problems in Data Communications Between the User and the Computer	L. A. HITTEL	395
Communications Needs of the User for Management Information Systems	D. J. DANTINE	403

Elementary Telephone Switching Theory Applied to the Design of Message Switching Systems	L. STAMBLER	413
A Proposed Communications Network to Tie Together Existing Computers	T. MARILL L. G. ROBERTS	425

SCIENTIFIC APPLICATIONS

The Lincoln Reckoner: An Operation-Oriented, On-Line Facility with Distributed Control	A. N. STOWE R. A. WIESEN D. B. YNTEMA J. W. FORGIE	433
Telsim, A User-Oriented Language for Simulating Continuous Systems at a Remote Terminal	K. J. BUSCH	445
Man-Machine Communication in On-Line Mathematical Analysis	R. KAPLOW J. BRACKETT S. STRONG	465

IMPACT OF COMPUTERS ON GOVERNMENT: FEDERAL, STATE, LOCAL

The Check Payment and Reconciliation Program of the U. S. Treasury	G. F. STICKNEY	479
Problems of Information Systems in State Governments	D. G. PRICE	501
Impact of Computers on Local and Regional Government	H. H. ISAACS	505
An Information System for Law Enforcement	L. B. MCCABE L. FARR	513
The Transfer of Space and Computer Technology to Urban Security	R. B. HOFFMAN	523

THE MAN-MACHINE INTERFACE

Recent Progress on a High-Resolution, Meshless, Direct-View Storage Tube	N. H. LEHRER R. D. KETCHPEL	531
The Plasma Display Panel—A Digitally Addressable Display with Inherent Memory	D. L. BITZER H. G. SLOTTOW	541

SELECTED APPLICATIONS USING NUMERICAL ANALYSIS

The Use of Semi-Recursive Polynomials in the Design of Numerical Filters	C. B. STALLINGS	549
Fast Fourier Transforms—For Fun and Profit	W. M. GENTLEMEN G. SANDE	563

HIGH QUALITY PAPERS OF GENERAL INTEREST

A System for Automatic Value Exchange	S. C. BLUMENTHAL V. F. HAKOLA	579
Real-Time Recognition of Handprinted Text	G. F. GRONER	591
Basic Hytran Simulation Language—BHSL	J. C. STRAUSS	603

ADVANCES IN PROGRAMMING LANGUAGES

A Processor-Building System for Experimental Programming Language	T. W. PRATT	613
The Introduction of Definitional Facilities into Higher Level Programming Languages	T. E. CHEATHAM, JR.	623
Foundations of the Case for Natural-Language Programming	M. HALPERN	639
Explicit Parallel Processing Description and Control in Programs for Multi- and Uni-Processor Computers	J. A. GOSDEN	651
The Lisp 2 Programming Language and System	P. W. ABRAHAMS	661
APL—A Language for Associative Data Handling in PL/I	G. G. DODD	677

COMPUTER-ORIENTED DATA ANALYSIS

Automatic Off-Line Multivariate Data Analysis	G. SEBESTYEN	685
Data Analysis and Statistics: An Expository Overview	J. TUKEY M. WILK	695

TECHNOLOGIES AND SYSTEMS FOR ULTRA-HIGH CAPACITY STORAGE

UNICON Computer Mass Memory System	C. H. BECKER	711
An Electron Optical Technique for Large-Capacity Random-Access Memories	S. P. NEWBERRY	717
A System of Recording Digital Data on Photographic Film Using Superimposed Grating Patterns	R. L. LAMBERTS G. C. HIGGINS	729
A Photo-Digital Mass Storage System	J. D. KUEHLER H. R. KERBY	735

HYBRID APPLICATIONS AND TECHNIQUES

Hybrid Computers in the Analysis of Feedback Control Systems	C. K. SANATHANAN J. C. CARTER L. T. BRYANT L. W. AMIOT	743
A Hybrid Computer Solution of the Co-Current Flow Heat Exchange Sturm-Liouville Problem	L. T. BRYANT L. W. AMIOT R. P. STEIN	759
A General-Purpose Analog Translational Trajectory Program for Orbiting and Reentry Vehicles	A. I. RUBIN L. SHEPPS	771
Satellite Lifetime Program	J. L. STRICKER W. W. MIESSNER	789
Trajectory Optimization Using Fast-Time Repetitive Computation	J. S. RABY R. C. WINGROVE	799
COMMITTEE LISTS		809
List of Exhibitors		817
Author Index		819

A CONVERSATIONAL SYSTEM FOR INCREMENTAL COMPILATION AND EXECUTION IN A TIME-SHARING ENVIRONMENT

James L. Ryan

Tymshare Incorporated, Los Altos, California

Richard L. Crandall

Com-Share, Incorporated, Ann Arbor, Michigan

and

Marion C. Medwedeff

Scientific Data Systems, Santa Monica, California

BACKGROUND

The Conversational Compiler System described herein is implemented on the Scientific Data Systems Model 940 Time-Sharing System. The SDS-940 has, as its Central Processor, a modified SDS-930, the modifications for which were developed at the University of California at Berkeley by Melvin Pirtle.¹ This hardware includes a paging scheme, a set of privileged instructions for monitor as opposed to user mode of operation, and the ability to perform input/output operations to secondary memory while computing.

The internal organization of CCS was motivated by ideas from Dr. Kenneth Lock² of the California Institute of Technology at Pasadena, California, and by B. Randell and L. J. Russell in their book, "ALGOL 60 Implementation."³ CCS operates as a subsystem of the System Monitor developed for the

SDS-940 by Dr. Wayne Lichtenberger, Butler Lampson,⁴ Peter Deutsch and Larry Barnes, all of the University of California at Berkeley. CCS itself is not involved in the basic management of the system input/output, physical memory allocation or scheduling of user programs. It does, however, take into account the physical restrictions of its environment such as page size, read only vs read/write memory, and so forth.

The Berkeley time-sharing system includes almost all of the service routines and other functions and controls that were required by the design of CCS. It was found that certain convenient hardware features were missing, but none was a serious hindrance to the design of the system or its compilers.

The design of CCS is not oriented towards any particular computer system; however, many of its features would be difficult to implement or would be totally ineffective without a favorable environment

such as that provided by the SDS-940 system and the Berkeley Monitor.

CCS and its compilers were developed at Tymshare, Inc. in Los Altos, California, in cooperation with Com-Share, Inc. of Ann Arbor, Michigan, and Scientific Data Systems of Santa Monica, California.

SYSTEM OBJECTIVES

The design of the Conversational Compiler System is predicated upon the attainment of several objectives; specifically:

Conversational Interface. Since CCS will be used in an on-line time-sharing environment, a high level of interaction between the user and the system is imperative for maintenance of operational efficiency. Therefore, the primary objective of the system design becomes that of establishing a practical means for dialogue between the user and the system.

Multiple Language Capability. It has become an accepted axiom in the computing field that no single programming language provides the best path to solution of all problems. Thus, CCS is designed to be multilingual. The initial implementation, as described in this paper, includes extended versions of full ALGOL 60 and FORTRAN IV.

Incremental Compilation and Execution. During the development of a program, there is likely to be an interleaving of executions and program modifications. To cope efficiently with this, compilers should be incremental and execution should be controllable. Incremental compilers would allow for fast turn-around time and minimal work for the system in keeping up with programmer modifications. Controlled execution would allow the user to specify ranges of statements to be executed as well as single statement step execution.

Language Oriented Program Debugging. Since the programmer should be ignorant of the idiosyncratic nature of the computer itself, the debugging techniques available should be in terms of the conceptual program rather than of the computer upon

which that program is executed. Debugging information should be provided, therefore, in a direct relationship to the source language statements in the user's program and the user should be able to easily obtain information informing him of the effect of individual statements. Furthermore, the programmer should also have explicit control as to the nature of the debugging information provided.

Multiple Mode Execution. Normally, following modification of a program, a trial execution is undertaken to determine whether the modification has the desired effect on the operation of the program. During this trial execution, a tight watch should be maintained and the programmer informed of any questionable condition that may arise. It is important to note that even though a condition may be questionable, the effect of this condition may not be adverse. Therefore, following this trial execution, if it has been determined that the program is in proper operating order, the programmer should have the capability of turning off these alarms. This will require a dual mode execution capability, one executing mode for program checkout and a second for production use.

Common Internal Format. The output of each of the language compilers should be in a compatible form so that programs may be comprised of segments written in different languages. This capability would make it possible, for instance, to use subprograms written in ALGOL with a control program written in FORTRAN.

Multiple Mode Program Storage. The programmer should have the capability of creating Save files of a program in either symbolic or internal forms, or a combined file that preserves both the symbolic and internal forms, so that program modification may be continued at a later time.

Re-Entrant Characteristics. Given the multiprogramming capability that accompanies the more advanced time-sharing systems, heavily used subsystems should be re-entrant. Time-sharing also introduces the

capability for several people at different remote terminals to operate simultaneously, perhaps sharing a user-written program. This implies that compilers should generate re-entrant code allowing the user to write shareable programs which are not a part of the computer facility library.

User Aid. A certain degree of self-teaching capability has already been given to conversational systems via rapid answer-back of error diagnostics. This ability should be extended to allow a more natural question and answer capability. The user should be able to ask English language questions about his problems and get immediate responses from the computer. This would replace the sometimes lengthy process of referencing a manual which never seems to be present when needed.

SYSTEM OPERATION

The Conversational Compiler System consists of a language independent supervisory program, service routines, and associated language compilers. CCS, itself, operates as a subprogram to a system executive. Upon receipt of the proper language identification command, the system executive activates CCS, which is then controlled through its own system command set.

CCS System Commands

Command Recognition. All CCS System Commands are of the form:

```
<command identifier>
  [, <specification field> ...]!
```

The command identifier may be abbreviated by including only enough characters to uniquely identify the desired command from the other commands in the command set. Also, as will be shown, the character content of the command identifier may be redefined by the user.

Statement Numbers. Every statement in a CCS program has a statement number used only for text manipulation and never for the executing logic of the program. Statement numbers are assigned either explicitly by the programmer or implicitly by the

system, as will be shown. Statement numbers are decimal numbers with a maximum of three integer and three fractional digits. Insignificant zeroes are not considered as part of the statement number. The smallest and largest statement numbers are .001 and 999.999, respectively.

Statement Designators. Most of the CCS System Commands use statement designators to enable the user to describe the portion of the program to be affected by the command.

Statement designators may take the form of:

- A statement number (+ and - may be used to symbolically represent the highest and lowest existent statement numbers).
- A statement number with increment (specifying a statement which precedes (negative increment) or follows (positive increment) the numbered statement by the number of statements indicated).
- An insertion indicator for the COMPILE command. (A statement number followed by a + or - sign indicating whether the insertion precedes or follows the numbered statement.)
- A statement range indicator (two of the above separated by a colon.)

Program Text Control.

```
COMPILE, <statement designator>,
        <in-statement>!
```

Enters the in-statement into the program as directed by the statement designator.

The in-statement may be one or more source language statements separated by the proper source language delimiters. Statement designators contained within a parenthetical pair and bounded by the appropriate statement delimiters may appear in the in-statement. These statement designators identify statements in the program that are to be included in the text of the in-statement.

A file name, identified by the surrounding quotes, appearing in an in-statement causes the symbolic text from the named CCS program file to be included in the in-statement.

```
DELETE, <statement designator> [, <state-
  ment designator> ...]!
```

Removes designated statements from the program.

RESEQUENCE, (<statement designator>, <statement designator>) [(<statement designator>, <statement designator>...)]!

Assigns new statement numbers to the statements identified by the first statement designator in the parenthetical pair as directed by the second statement designator which may specify an allowable statement number range, or a base number and an increment.

LOCK, <statement designator> [, <statement designator>...]!

Declares a protected status for the designated statements. Locked statements cannot be removed from the program, or be renumbered. Copying does not affect the lock status.

UNLOCK, <statement designator> [, <statement designator>...]!

Removes protected status from designated statements.

OFF, <statement designator> [, <statement designator>...]!

Temporarily deactivates designated statements, removing them from the executing logic of the program.

ON, <statement designator> [, <statement designator>...]!

Reactivates designated statements, restoring them to the executing logic of the program.

Program Execution.

```
EXECUTE [, <statement designator>]
      [ ,DIAGNOSTIC ]!
      [ ,PRODUCTION ]
      [ ,STEP ]
```

Executes the designated statements according to the specified mode. (If no mode is specified, PRODUCTION is assumed.)

DIAGNOSTIC—causes a pause and output of an appropriate message whenever an error or questionable condition occurs.

PRODUCTION—Aborts program when-

ever an error occurs. Questionable conditions trapped in DIAGNOSTIC mode will be ignored.

STEP—Same as DIAGNOSTIC except that a pause occurs after completion of each statement (and its debugging routine) allowing another system command to be entered or execution to be resumed (by entering an exclamation point).

CLEAR!

Destroys internal data storage left after a partial execution.

PERFORM, <in-statement>!

Executes the in-statement, but does not enter it into the program. The in-statement may modify, but not declare, data storage.

Program Debugging.

DEBUG, (<statement designator>, <statement designator>) [, (<statement designator>, <statement designator>)...]!

Declares statements identified by the second statement designator within each parenthetical pair to be used as a debugging procedure for the statements identified by the corresponding first statement designator. The debug procedures will be executed immediately following each statement for which they are declared. It is possible for more than one debug procedure to be declared for a given statement.

Program Information.

LIST [, <statement designator>...]

```
[ , QUICK]!
[ , FORMATTED]
[ , LOCK]
[ , UNLOCK]
[ , OFF]
[ , ON]
[ , DEBUG]
[ , ERROR[S]]
```

QUICK — Prepares high-speed unformatted listing.

FORMATTED — Prepares formatted listing in which statements are indented according to program structure.

LOCK, UNLOCK, OFF, ON, DEBUG,

ERROR[S] — List only statements of the designated type.

The directives may be used in combinations. If no directives appear QUICK is assumed.

FIND, <label>!

Lists the first occurrence of the indicated source language label. Subsequent occurrences of the same label may be found by entering an exclamation point.

File Control.

LOAD, <file-name>!

Loads the specified CCS program file.

SAVE, <file-name> [, SYMBOLIC]!

Saves the program on the designated CCS program file. If indicated, only the symbolic text will be saved.

System Control.

DEFINE, <old command identifier> = <new command identifier>!

Specifies new character set to be used for command identification.

Diagnostics

CCS provides three categories of diagnostic messages; command, compilation and execution. An immediate warning that an error has occurred is given (for the teletype, a bell) so that the current activity may, at the user's option, be interrupted for corrective action. Depending on the nature of the problem, however, immediate service may not be required. For instance, during compilation of program text, even though a warning occurs upon detection of each syntactical error, correction of the errors may be postponed until completion of the input of the text.

Command Diagnostics. Command diagnostics, given in the form of a repeated warning, occur whenever the intent of a command is not understood. In these instances, the command should be re-entered.

Compilation Diagnostics. Whenever syntax errors in program text are detected, an immediate warning is given. Upon the completion of the processing of all statements in the in-statement, the diagnostic information is returned as illustrated below:

User:

```
COMPILE, 5:10, A := B + C; X :=
  Y * / Z; P = Q + R!
```

System: 2 SYNTAX ERRORS

User: LIST, ERRORS!

```
System: *7. X := Y * / Z;
```

```
          ↑
*9. P = Q + R;
          ↑
```

The asterisk preceding the statement number indicates the statement is a part of an in-statement not yet structured into the program. The position of the upward arrow locates the position in the statement where the error was detected. No further syntax scan was made of the erroneous statement. If this is an insufficient description of the error, the query WHY may be used to obtain a detailed explanation as follows:

User: WHY, *7!

```
System: *7. BINARY OPERATOR NOT
PRECEDED BY VARIABLE OR CON-
STANT.
```

If errors occur in an in-statement, corrective action *must* be taken in the form of statement deletion or replacement. New statements will not become part of the program until all errors in the in-statement have been removed. COMPILE and DELETE commands may be used to modify the in-statement.

In the above example, statements *7. and *9. were in error. The commands:

```
COMPILE, *7 : *9, X := Y * 5 / Z;
P := Q + R!
```

```
COMPILE, *5: *10, (6); (1); X :=
Y* 5/Z; (*5)!
```

```
DELETE, *7 : *9!
```

would all be valid corrections. These commands modify the in-statement group. Note, in the second example above, that statements previously compiled, either within or without the in-statement designator range, may be copied into the in-statement. This modified in-statement replaces the original in-statement. Since it is possible that other errors still exist or new errors have been introduced, the error search is repeated. Error correction proceeds until the entire in-statement is free of errors. The in-statement is then placed into the program as indicated by the

statement designator of the original COMPILE command.

Execution Diagnostics. There are two forms of execution diagnostics. Those that occur only when the program is being run in DIAGNOSTIC mode and those that occur regardless of the mode of execution. If an execution diagnostic appears and the program is in DIAGNOSTIC mode, a pause occurs. If the diagnostic condition is to be ignored, execution may be continued. Execution diagnostics appearing while the program is in PRODUCTION mode terminate the program.

CCS User Aid

Provided with CCS is an on-line question answering service called HELP.* HELP accepts English language questions about CCS, ALGOL, or FORTRAN. The questions are analyzed by a key word search; various common endings on key words being viewed interchangeably (e.g., the word COMPILING looks the same as the word COMPILE). The best answer is chosen for the question and is printed out.

To use HELP, the user merely types: HELP!

as a command to CCS. The system is then set to accept questions.

For example:

User: HELP!

User: WHAT IS A STRING LITERAL?

System: A STRING IS A SERIES OF CHARACTERS PRECEDED BY A PRIME (') AND FOLLOWED BY A QUOTE ("). E.G., 'STRING'

User: HOW DO I GET A PRIME IN A STRING?

System: STRINGS MAY BE NESTED WITH THE CONVENTION THAT ANY STRING NESTED WITHIN ANOTHER STRING HAS ITS SURROUNDING PRIME AND QUOTE INCLUDED IN THE SURROUNDING STRING. E.G., 'A'B'C' HAS THE CONTENTS A'B'C.

User: HOW DO I GET ONLY A PRIME WITHOUT A QUOTE IN A STRING?

System: USE NESTING AND THEN EXTRACTION.

* Actually the HELP system for CCS is an extension of the Question Answering System (QAS) developed at the University of California, Berkeley.⁵

User: HOW DO I EXTRACT CHARACTERS FROM A STRING?

System: THE SYSTEM ROUTINE LEFT (N, S) RETURNS THE STRING WHICH IS THE LEFT N CHARACTERS OF STRING S. THE SYSTEM ROUTINE RIGHT (N, S) RETURNS THE RIGHT N CHARACTERS OF S.

Edit Features During Text Input

Editing features are available to the user while entering program text.

The symbol # deletes the preceding character. This may be used iteratively but applies only to the current statement of input text.

The symbol ← deletes the current statement.

A string of characters between two statement delimiters may be considered to be a text segment.

The formation:

—*n*:

appearing at the beginning of a segment replaces the text segment *n* segments previous with the text segment following the colon. If a statement delimiter immediately follows the colon the *n*th previous segment is deleted. If *n* refers to a segment not in the in-statement, the text segment following the colon is inserted at the beginning of the in-statement. *n* may not be zero.

Example:

```
COMPILE, 1 : 3, A : =B+*C##C; Q : =
E/*; C : =Q-F; -2 : Q : =E/N;!
LIST, 1 : 3!
```

1. A : =B+C;
2. Q : =E/N;
3. C : =Q-F;

SYSTEM ORGANIZATION

System Executive. The Conversational Compiler System consists of a group of programs which operate as a subsystem to a System Executive. Control is given to CCS by the System Executive upon receipt of an appropriate command, there being an entry for each implemented programming language. Although CCS is intended for use in a time-sharing environment, most time-sharing attributes are primarily a function of the System Executive, which of itself is not a topic of this paper.

Command Dispatcher. CCS operations are controlled through a Command Dispatcher which interprets all system commands. Upon command identification, control is transferred to an appropriate system routine.

Source Language Compilers. The initial implementation of CCS has two language compilers; extended versions of ASA FORTRAN IV and ALGOL 60. The compilers are entered from the Command Dispatcher, and produce independent relocatable program elements for each segment of source language.

Source Language Structuring Routines. Upon the completion of the compilation of a group of source language statements, control transfers from the compiler to the appropriate language structuring routine. Whereas the compilers operate incrementally, the structuring routine interacts with all of the elements produced during a compilation phase, linking these elements into a meaningful "structure" and entering it into the program. Upon completion of this task, control returns to the Command Dispatcher.

Source Language Definition Search Routines. Although the structuring routines link statements into a program, they do not supply operating definitions needed for execution. These definitions are supplied by the appropriate language structuring routine which is called from the Execution Monitor upon recognition that a particular program element

has not been previously executed. After definitions have been supplied for this element, execution proceeds in a normal fashion.

Execution Monitor. The Execution Monitor controls program execution, its action being determined by the specified execution mode, i.e., PRODUCTION, DIAGNOSTIC, or STEP. The Execution Monitor may interrupt the program upon the completion of a statement; pausing or terminating the execution if ordered. Upon completion of the execution of a program statement, the Execution Monitor will execute any debugging routines active for the respective statement.

Service Routines. These routines perform the necessary functions of preparing program listings, changing program statement status, destroying data following partial execution, deleting program text, preparing program files, and defining debugging routines.

SYSTEM MECHANICS

Memory Organization

The total computer memory allocation for each CCS user can be divided into two general classifications; System Memory, which is shared by all users, and User Memory, which is unique in content for each user.

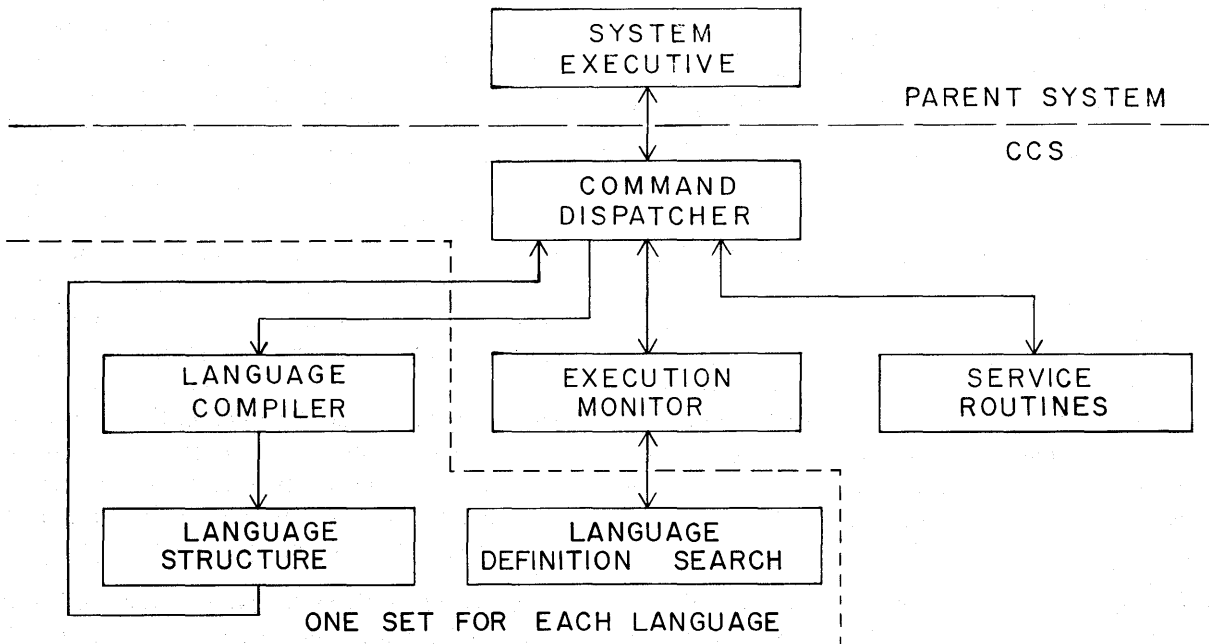


Figure 1. CCS system organization.

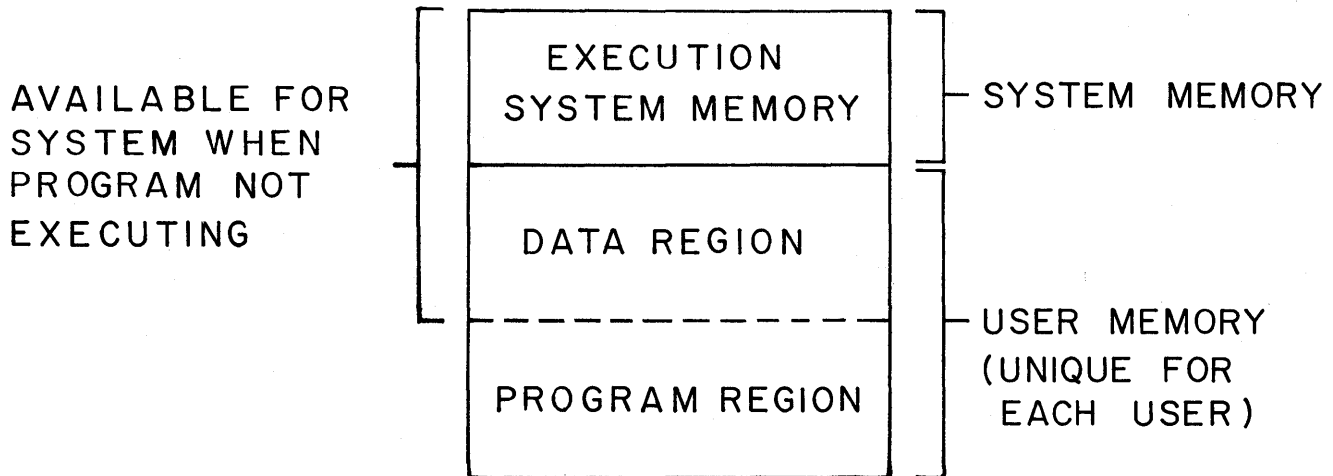


Figure 2. Memory organization scheme for CCS user.

System Memory. The proportion of the total memory commitment given to system use varies with the specific task being undertaken. Even though CCS is comprised of many routines, only those needed to perform current functions are included in the user's allocation.

User Memory. User Memory is always separated into two sections; the program region, and the data region. The program region is used to store the program itself, and acts as a de-facto communication buffer between the compilation, structuring, and definition search routines.

The data region is used during program execution as the working area for data manipulation and storage. During a non-executing phase, the data region may be swapped for system memory allowing more efficient use of the total memory commitment.

From Source Language to Object Program

Compilation. All output from the CCS Language Compilers is placed into the program area of user memory. This output consists of a threaded list of "elements," where the actual placement of each element is determined by the compiler upon inspection of a "hole" list. Each of these elements contains the encoded representation of a source language statement, directive information for the structuring routine, or diagnostic information describing a syntax error.

Structuring. Upon the completion of the compilation of a given in-statement, the threaded list of elements produced become, in effect, "data" for the appro-

priate language structuring routine. The first task is to process elements indicating that other elements from the original program are to be "copied" into the element list. Following this task, the elements are counted in order to determine the increment to be used for calculating statement numbers. The list is then searched for diagnostic elements and if any are found an appropriate message is output and action taken which causes the system to expect corrective measures from the user. These corrective measures modify the element string. Following modification, the diagnostic search is again undertaken.

When no more diagnostic elements remain in the element list, the formal structuring process begins. Each element in the list is, in turn, checked for type and it is determined if the statement has a proper relationship to the other statements in the list. If an illegality is detected, the action taken is the same as that taken for syntax errors. If no errors are detected, a new header is created for each element containing the "structure pointers" which define the relationships between the different elements.

As part of the structuring process, modifications are made to the Symbolic Text Dictionary which contains one entry for each statement in the program. Dictionary entries are threaded in the order in which the statements appear in the text of the program. Each entry contains a pointer to the corresponding text on the Symbolic Text File and the core location of the element representing the statement.

First Execution. Even though the structuring routines connected elements into the program, complete definitions for the program variables were not yet

supplied. This task is undertaken by a "definition search" routine. Upon encountering an element which has not been previously executed, the Execution Monitor causes a definition search to be activated which traces through the program seeking definitions for the undefined variables. If a definition is not found, depending on the conditions and the source language used, one of two actions will be taken. An execution diagnostic will be output, or an implicit declaration for the variable will be entered into the program.

Subsequent Execution. Once definitions are supplied via the first execution, an element may be executed any number of times without further ado. Changes to the data definitions or placement of the statement in the program may, however, require another definition search.

Pseudo Machine Code

The encodation produced by CCS translators is in the form of a quasi-interpretive language referred to as "Pseudo Machine Code" or PMC. The net effect of PMC is the expansion of the "hardware" capabilities of the computer, thereby simplifying the task of the compiler and enhancing the capabilities of the system.

Data Formats. The PMC allows internal data to assume one of seven formats:

- Integer
- Real
- Boolean
- Expanded Precision Real
- Complex
- String
- Text

Text and string variables have identical functions, the difference being that while the storage for string variables is taken from the user's memory, the storage for text variables will be taken from secondary memory. The string and text variables are unique in that they store character strings of indefinite length. The amount of storage used is not fixed, but rather, is a product of the number of characters actually contained in the string. Therefore the storage used by string or text variables will possibly change during program execution.

Any of the PMC data types may be dimensioned to be arrays. Also, PMC provides the capability for

function subroutines to be defined in terms of any of the above data types.

Instruction Set. PMC operations may be grouped into three categories:

- Data Operations
- Storage Declaration
- Procedure Control

Data Operations. PMC data manipulating instructions are provided for arithmetic, relational and Boolean operations. These instructions function independently of data types.

Depending on the mode of execution, illegal data operations produce either an undefined result or an execution diagnostic.

Storage Declaration. PMC storage declarations take the form of a subroutine call with argument list. When encountered during program execution, appropriate modifications are made to the data base conforming with the intent of the declaration.

Procedural Control. Within the PMC instruction set, operators are supplied for the following procedural control capabilities:

- Transfer of Control
- Subprogram Call with Parameters
- Loop Control

Transfers of control may be either conditional, unconditional, or selective. A conditional branch is made or not made depending upon the content of a Boolean variable. Selective branches may be made to one of many locations depending on the resultant value of an expression.

PMC provides a complete mechanism for subprogram calls. Parameters may or may not be present, and the parameters themselves may effect the transfer of a value or identifier into the subprograms. The parameters may, if desired, be expressions, and it is possible to indicate whether the expression should be reevaluated upon each use of the corresponding entry in the subprogram, or whether the expression evaluation is undertaken only upon transfer of control to the subprogram.

The mechanism for programmed loops has been provided in the Pseudo Machine Code. This mechanism is flexible enough in scope to be capable of proper performance of all variations of ALGOL "for" statements and FORTRAN "do" statements.

Data Stack. The executing logic of the PMC assumes a data stack. At the beginning of program execution,

the stack is empty. Data storage declarations cause the necessary space to be placed on the top of the stack (the PMC includes operations for subsequent alteration of the size of the storage area). The top of the stack is used as a scratch pad by the PMC representing a given statement. The executing PMC access data on the stack through a network of dynamically maintained indirection pointers. This mechanism allows data definitions to be recursive since the obscured definitions can be restored when needed.

CCS INPUT/OUTPUT PACKAGE

A flexible and easy to use I/O package is available for CCS users. Implicitly formatted I/O capability is offered for easy data transfer to and from a character oriented remote terminal. Explicitly formatted I/O is available for more elaborate data input and output to and from any storage media. Formatted I/O is not device oriented, and only a file designator (a system assigned number) is needed to reference the file.

Format specifiers are designed for both business and scientific applications. Additional flexibility is obtained by using string data for formats. Through the use of powerful string handling capability available to the CCS user, formats can be constructed, or modified, during program execution.

In order for all CCS languages to have ready access to the I/O package, commands take the form of system subroutine calls.

Implicitly Formatted I/O

The subroutine:

ACCEPT (*vl*)

(where *vl* is a list of variables, possibly subscripted, separated by commas) will input data from the teletype. A space, comma, or carriage return serves to separate data. Data is input until the *vl* is exhausted. Strings as well as numbers may be input in this fashion using the prime and quote as string delimiters.

The subroutine:

DISPLAY (*el*)

(where *el* is a list of general expressions separated by commas) will output data to the teletype. Each value that is printed is followed by three spaces. If a string is printed, these spaces are suppressed. No carriage return is issued until the *el* is exhausted, or an output line is filled, at which time a carriage return

and line feed are automatically given. The terminating carriage return may be suppressed by including the system defined symbol "\$" as the last element in the *el*. "%" may be used to obtain intermediate carriage returns; e.g., ('A=', %, 3) would print:

A=
3

Implicitly formatted I/O is intended to be a painless way for novices and impatient experts to communicate rapidly with their programs.

Explicitly Formatted I/O

The subroutine:

READ (*n, f, vl*)

inputs data from the file *n* according to the format *f* into the variables listed in *vl*. *f* may be either a string literal or variable.

The subroutine:

WRITE (*n, f, el*)

outputs the values of the expressions in *el* to file *n* according to format *f*.

Formats

A format is a string which is either in literal form or addressed by a string variable. Each element in the I/O list (i.e., *vl* or *el*) for the READ and WRITE routines is described by a format part. Parts are separated from each other by a colon (e.g., 'part : part : part').

A part consists of special characters which describe input and output fields.

A string of duplicate characters may be expressed also with the number *n* preceding the repeated character; e.g., DDDD is equivalent to 4D.

Any part may be enclosed within parentheses and preceded by a number *n* indicating a concatenation of the part; e.g., '2(DD.D :)' is equivalent to 'DD.D : DD.D'.

When a format string is exhausted before the I/O list has been completely processed, the format is rescanned.

All CCS I/O routines count lines and space out pages on teletype I/O. Symbolic files have the necessary line feeds such that if they are ever printed, spaced pages result.

When a specific field size is exceeded by trying to output too large a number, an asterisk prints in the leftmost position of the field along with the remaining part of the number that will fit.

Any string literal appearing within a format is printed verbatim excluding the surrounding primes. String literals appearing in a format are ignored when input, allowing an output format in many instances to be used for input.

If a real (integer) datum is input and the receiving variable is integer (real), the necessary conversion to real (integer) takes place. This implicit conversion feature extends to strings.

Boolean values are printed as a "T" or "F" right justified in the field. Any format character specifying digit may be used for Boolean output.

Whenever an I/O list is exhausted, a carriage return and line feed are automatically given. A special character described below may be used to suppress this action.

Special characters:

D : Specifies a digit for the corresponding position in the I/O field. Insignificant zeroes are printed and the sign is suppressed;

e.g., the Format: 'DDD'
 used with: Write: 3
 will Print: 003
 used with: Write: -3
 will Print: 003

+ : Same as D except the sign "floats" into the rightmost "+" which is not needed for a significant digit;

e.g., Format: '+ + D'
 Write: 5
 Prints: ^ + 5
 Write: -5
 Prints: ^ - 5

- : Same as "+" except that a plus sign is never printed;

e.g., Format: '- - D'
 Write: 5
 Prints: ^ ^ 5
 Write: -5
 Prints: ^ - 5

Z : Same as "D" except leading zeroes are not printed;

e.g., Format: 'ZZZ'
 Write: 6
 Prints: ^ ^ 6
 Write: -6
 Prints: ^ ^ 6

, : May appear anywhere in a part and causes the printing of a comma. The comma is not printed with an adjacent space;

e.g., Format: 'ZZZ,ZZZ,ZZZ.DDD'
 Write: 532468.29
 Prints: ZZZ 532,468.290

The symbols "D", "+", "-", "Z", and "," may be used for input of integer data. The datum must be such that it could have been created via a WRITE using an identical part specification.

. : The decimal point indicates a real number is to be output. The special characters to the left of the decimal point describe the integer portion of the datum while the fractional portion is described by the characters to the right. The decimal point prints in all cases except when surrounded by blanks. The definitions for "+", "-", and "Z" are mirrored when to the right of the decimal point;

e.g., Format: 'ZZZ.ZZ'
 Write: 48.7
 Prints: ^ 48.7.
 Format: 'ZZZ.D + +'
 Write: 63.4
 Prints: ^ 63.4 +.

@ Represents base 10 exponentiation.

E: The coefficient is described by the characters to the left of the @ or E symbol and the exponent is described by the format characters to the right;

e.g., Format: 'ZZZ.Z @ DD'
 Write: 463.2
 Prints: 463.2 @ 01

Real numbers are input by specifying their field with the "D", "Z", "+", "-", ".", ",", "@". The number is automatically converted to the mode of the corresponding variable in the input list;

S : Specifies a scale factor to be used after input or before output of numeric data.

The formation:
 S[n]
 appearing at the beginning of a format

part (n is an integer, possibly signed) scales the number by 10 to the power n ;

e.g., Format: 'S[-2]DD"
 Write: 3000
 Prints: 30
 Format: 'S[2]DDD"
 Write: 4
 Prints: 400

X : Acts as a skip on input and as a space on output. The "X" may occur anywhere in a format part;

e.g., Format: 'DD : XX : DD"
 Write: 23,22
 Prints: 23 ^ ^ 22
 Format: 'ZXZZ"
 Write: 468
 Prints: 4 ^ 68

L : Skips to the next line on output; % ignored on input.

P : Skips to the next page on output; ignored on input. After the page skip, "L" is implied.

A : Specifies alphanumeric input or output;

e.g., Format: 'AAA : ZZ"
 Write: 'A ^ =', 2
 Prints: A ^ = ^ 2

I : Specifies implicit formatting; causing READ to function as an accept and WRITE to function as a DISPLAY. The symbol "I" may not be used in conjunction with any other special characters other than "S".

The format specifier "I" is useful for input of strings of unknown length. Strings input must have primes as delimiters. Output strings will be printed with a special "invisible delimiter" which will not print, but will appear on a paper tape or file, allowing these strings to be input without apparent primes.

\$: Indicates a binary file is to be input from or output to. No other specification characters may appear in the format.

B : Suppresses the carriage return when the I/O list is exhausted, allowing more than one output statement to print on a given line.

CCS-ALGOL

The motivation for creating CCS-ALGOL stems from the desire to create, in conversational form, an ALGOL 60 that adhered to the specifications of the 1962 Revised Report.

Much has already been said about the effects of time-sharing on software. The one very important effect of time-sharing is the fact that it enables the user to operate a computer on-line. He therefore expects things to happen very rapidly. He also expects to have reasonable conversations with the machine in order to debug programs and operate them successfully. This necessitates a responsiveness to the user's commands in terms of seconds rather than minutes (or hours) as in batch processing oriented compilers. CCS-ALGOL has, in large measure, achieved this goal by compiling input text in an incremental fashion. A high degree of interaction between CCS-ALGOL and the CCS Execution Monitor allows rapid repair of bug-infested programs.

Another goal was to have the object code produced by the compiler, operate in a reentrant fashion, allowing users to write programs and subsequently share them simultaneously with other users.

Finally, some rules defined under ALGOL 60 having to do with expression formation, name formation and the use of strings were thought to be too restrictive and were somewhat relaxed.

The philosophy adopted during the development of the compiler, whenever a choice existed between different methods of implementation, was to choose that which allowed the greatest syntactic flexibility and ease of programming.

The above goals have been attained in the implementation of CCS-ALGOL. CCS-ALGOL is a fast, incremental, one-pass reentrant compiler producing reentrant object code.

A review of the more important capabilities and extensions of CCS-ALGOL follows. The general manner in which the compiler works is subsequently described. It is assumed that the reader is familiar with ALGOL 60.

Definitions

Identifiers. An identifier is a programmer-defined name used to represent variables, arrays, procedures, switches and possibly labels. An identifier is defined to be any string of alphanumeric characters of arbitrary length, containing at least one alphabetic character. Since blanks are ignored, the name "ARRAY BETA" is the same as the name "ARRAYBETA."

Examples: ABC, VARIABLE1,
THIS IS A NAME, 123Q4

Labels. A statement label may take either the form of an identifier or of an integer numeric. Insignificant zeroes on numeric labels are ignored; e.g., 0068 = 68.

Examples: 12345, AB1, 015,
THIS IS A LEGAL LABEL,
4368J, 236

Delimiters. ALGOL 60 requires certain words which have special meaning such as "GO TO" to be used as unique symbols in the character set. These special symbols and other single character symbols such as arithmetic operators, are called delimiters. While delimiters such as =, +, ↑, >, have unique representations on currently available input equipment, delimiters such as "IF" and "GO TO" do not, and in order to set them apart from identifiers (e.g., a variable called "IF") CCS-ALGOL requires a single dot (.) to precede an alphabetic delimiter. Examples are: .IF, .GO TO, .ARRAY, etc.

In order to save typing, any delimiter comprised of five or more characters may be contracted by typing the dot, the first letter, a prime, and the last letter. The only exception to this rule is .BEGIN which has no abbreviation. For example, .PROCEDURE looks like .P'E and .BOOLEAN looks like .B'N.

Variables and Constants. Variables may be any of five types: text, string, Boolean, integer or real. A variable or array element not containing data of one of these types is considered undefined.

The Boolean constants are: .TRUE, and .FALSE, and are the only values to which Boolean variables or array elements may be set.

Integer constants have the appearance of a numeric string.

Real constants have the form:

$x.y$ or $a.b@c$.

x , y , a , b and c are integers. x , a and c may be signed. @ symbolizes "ten to the power". If $a.b$ is missing, 1.0 is assumed.

String constants have the form:

'STRING OF CHARACTERS'

Strings. Several string management features have been included in CCS-ALGOL. In addition to those described in ALGOL 60 specifications, a variable may be declared to be of type "string" by the new delimiter .STRING.

The declaration form:

.STRING .ARRAY $x[a_1:b_1, \dots, a_n:b_n]$

declares x to be an "n" dimensional array of strings. String array elements can contain strings of arbitrary length and, therefore, may be thought of as extending into the $n + 1$ st dimension.

Procedures may also be of string type in which case they are expected to return a string when called upon.

The relational operators (>, >=, <=, <, +, <>) may have string variables as both operands with the obvious meaning.

A string conversion capability is built into the replacement operator. Just as ALGOL 60 converts a value of real (integer) type to a value of integer (real) type across the := operator, a string may be converted to a real or integer number or vice versa.

Example: .STRING S; .REAL R;
.INTEGER I;
S := '463.29';
R := I := S

results in the real value 463.29 being placed in R and the integer 463 being placed in I.

Concatenation of two strings is performed by the system subroutine "JOIN" as follows:

$a := \text{JOIN}(s1, s2)$

where a , $s1$, $s2$ are string variables; $s2$ is appended to $s1$ and placed in a .

Extraction from strings is performed by the two system subroutines "LEFT" and "RIGHT" as follows:

$a := \text{RIGHT}(e, s1)$

Here the rightmost e characters of $s1$ will be placed in a .

Examples:

1. If $s1 := 'ABCDE'$
Then: $a := \text{RIGHT}(3, \text{LEFT}(4, s1))$
Results in: 'BCD' = contents of a .
2. If $s1 := 'ABCD'$
Then: $a := \text{JOIN}(\text{LEFT}(2, s1), \text{RIGHT}(1, 'EFH'))$
Results in: 'ABH' = contents of a .

NOTE: with proper comments, these subroutines are readable:

```
a := LEFT (3) CHARACTERS OF:
      (STRING 1)
q := JOIN (STRING 1) WITH: (STRING
      2)
```

Program Structure

CCS-ALGOL treats compound statements and blocks in an identical manner. Blocks may be nested to a depth of 64 levels.

Storage allocation is performed dynamically and, therefore, storage for identifiers declared in a particular block exists only during the execution of that block. Upon entrance to a block, storage is created for variables and arrays after evaluating subscript bounds for array declarations. Storage is released upon exiting the block within which it is defined.

Own variables and arrays have the special property that their storage is not released. However, storage for own arrays is dynamically allocated and array bounds may be reset by re-entering the block containing the declaration. If the new bounds are different than the previous bounds, elements of the previous array within the new bounds retain their previous values and subscripts. Array element values outside of the new bounds are permanently lost, while new elements have undefined values.

Statements and Expressions

The formation and meaning of all statements and expressions in CCS-ALGOL patterns that of the 1962 revised ALGOL Report, except for the following generalizations:

The IF Clause. ALGOL 60 specifies no .IF clause is to follow directly a .THEN delimiter because of parenthetical ambiguity. CCS-ALGOL adopts the convention that any .ELSE delimiter appearing in the text is associated with the nearest previous .IF delimiter that is not already matched with a .THEN.

Parentheses may be used to alter this precedence rule.

Example:

The statement:

```
.IF A .THEN .IF B .THEN .GO TO
L1 .ELSE .GO TO L2 is equivalent to:
```

```
.IF A .THEN (.IF B .THEN .GO
TO L1 .ELSE .GO TO L2) and to obtain
the alternate meaning, parentheses must
be used as follows:
```

```
.IF A .THEN (.IF B .THEN .GO TO
L1) .ELSE .GO TO L2
```

The Unary Operators. ALGOL 60 specifies that:

```
A .AND .NOT B
```

is a legal expression, whereas:

```
A*—B
```

is not. Since this is felt to be an unnecessary restriction, CCS-ALGOL allows all unary operators to appear without preceding parentheses. Unary operators may be repeated without ambiguity, although there appears to be no reason for doing this under normal programming conditions.

Example:

The expression:

```
A* — B + — — C
```

is equivalent to:

```
A* (—B) + (—(—C))
```

The Replacement Operator. The replacement operator “:=” may be used more than once in an expression to set more than one variable to an expressed value. The ALGOL 60 restriction that all identifiers in the left part of an expression be of the same type has been removed. If a mixture of variable types appear in the left part list appropriate mode conversions will be made.

Procedures. CCS-ALGOL allows a varying number of parameters to be furnished in different calls to the same procedure through the use of the new .LIST declarator which applies to the last parameter in the formal parameter list of a procedure definition.

Example:

```
.PROCEDURE PROC (A,B);
  .INTEGER A; .LIST B;
```

B defines a list, references to which must be subscripted. The convention is that B[n] in the procedure body references the n-1st element beyond the element corresponding to B in the calling sequence. If B[n] references an element which is not present (i.e., n too large in the call), B[n] becomes undefined.

Example:

Given the procedure definition:

```
.PROCEDURE PROC (A,B);
  .INTEGER A; .LIST B;
  B[A-1] := B[A];
```

The call:

```
PROC (2,Q,2)
```

would result in the following action: "A" has the value 2, therefore, B[A-1] or B[1] refers to Q, and B[A] refers to 2. PROC will, therefore, store 2 into Q.

For an array passed through a call in this manner, the following, rather awkward, notation results:

```
ARRAY [<list index>] [<array
subscripts>]
```

Procedures may recur, formal parameters being saved and restored accordingly.

When an expression appears as a parameter in a procedure call, references to the corresponding formal parameter in the procedure cause re-evaluation of the expression at the level in which the call appeared. The expression takes the form of an implicit subroutine eliminating duplication of in-line code in the procedure body.

Passing a procedure name through a procedure call, mentioning the corresponding formal parameter is tantamount to a call to that procedure.

Example:

Given the code:

```
.PROCEDURE PROC 1(A,B,C);
  A(B,C);
  .PROCEDURE PROC 2(M,N)
  M := N;
```

and issue the statement:

```
PROC1 (PROC2,Z,3);
```

The call to PROC1 causes the execution of the statement A(B,C) and, since PROC2 corresponds to the formal parameter A, a call to PROC2 results. PROC2 sets M := N or B := C, or Z := 3 and returns. PROC1 then returns completing the process.

A procedure name appearing in a calling sequence to another procedure will be executed every time the corresponding formal parameter is mentioned.

Example:

To perform the same action as the above:

Given the code:

```
.PROCEDURE PROC1(A);
  A;
```

```
.PROCEDURE PROC2(M,N);
```

```
  M := N;
```

and issue:

```
PROC1(PROC2 (Z,3));
```

The parameters supplied to PROC2 are supplied directly in this case.

When a statement label is furnished in a calling sequence, a transfer to the corresponding formal parameter effects transfer to the label.

Note that CCS-ALGOL allows numeric labels. The ambiguity problem posed by having labels which look like constants is solved by introducing the capability of maintaining doubly defined symbols using machinery transparent to the user. Suppose the constant 11 were used in a program in which there existed a label 11. Passing this numeric through a procedure call causes no difficulty in interpretation to CCS-ALGOL. A subsequent use of the corresponding formal parameter in an expression causes the value 11 to be used. However, a transfer to the formal parameter properly causes a transfer to the statement labeled 11.

External procedures may be coded in CCS-ALGOL or some other language translated separately—perhaps, not even under CCS. The only restrictions imposed are: (a) that communication with the call is through the formal parameters and, (b) these parameters may not include implicit subroutine calls.

All the standard system procedures specified by the report are available in addition to the previously mentioned I/O and string routines. These system routines have the property that if their name appears in a declaration, the name loses its system meaning and adheres to the user definition for the duration of that block. Upon exit from the block, the name regains its system meaning. There are, therefore, no names off limits to the user in CCS-ALGOL.

Implementation

The ALGOL Compiler is implemented in a time-sharing environment and has therefore been made to be modular and flexible. Full advantage is taken of the time-slice and paging characteristics which are becoming typical of time-sharing systems.

The compiler is comprised of two parts. The Text Scanner performs on-line editing and text manipulation. The Syntax Analyzer generates code on the basis of the meaning of the incoming program text.

The Text Scanner. The edit capabilities during text input have been described previously. A special pass must be made over the text to perform the edit functions, expand abbreviations and replace names and delimiters with their assigned internal codes.

The scanner operates on a line of text at a time. The text is scanned interpretively for edit characters until the line is clean. Text is then divided into segments to be written on the symbolic file according to a few rules. The rules are needed because the user is given the capability of deleting and changing "statements" as described under the CCS-command set. For instance, in ALGOL, a large block of code may comprise one compound statement which is normally looked upon as the same entity as a simple statement. However, the user would like not to have to delete the whole block (as a statement) in order to delete a single statement within the compound statement.

The rules that are followed are:

1. Simple statements terminate with a semicolon.
2. Statements within blocks terminate with either a semicolon or .END.
3. Statements within .IF statements terminate with .ELSE.
4. A .BEGIN terminates the preceding portion of the statement.

A statement such as: .IF B .THEN .BEGIN A := C; Q := D .END .ELSE G := F; would be broken down into the following text segments.

*Terminating
Delimiter*

- | | |
|----------------|------------|
| 1. .IF B .THEN | BEGIN |
| 2. A := C | ; |
| 3. Q := D | .END .ELSE |
| 4. G := F | ; |

The terminating delimiters are not included with the text. Instead a code is carried along indicating the terminating delimiter.

The above procedure has been adopted to make the DELETE work on meaningful statements in ALGOL. Actually, by noting the text segments above, it is seen that this entire process is transparent to the user. He merely treats blocks as a conglomerate of statements, and .IF statements as a triumvirate of statements.

The text segments are written into a symbolic file on secondary memory and are pointed to by elements in the text dictionary.

The text is then broken down into sections without regard as to where a statement begins or ends. A section is called an Identifier-Delimiter pair (I-D pair) and consists of either a delimiter or an identifier and a delimiter.

Example:

```
Text: |A :=| .IF| Q>|B. THEN| C
      .ELSE| D;| Q :=| M;|
      I=D Pairs:  A      :=
                  .IF
                  Q      >
                  B      .THEN
                  C      .ELSE
                  D      ;
                  Q      :=
                  M      ;
```

Each identifier is searched for in a name table. If found, its position number is returned. If not, a new entry is made and a new position number is created and returned. Delimiters are looked up in a delimiter table and replaced by their corresponding codes. A zero is used to fill the identifier position for those I-D pairs which contain only a delimiter. The I-D pairs are then loaded into a ring buffer in a buffer page.

The name table resides in the same memory page as does all buffer storage. This coexistence will work if the number of user defined names stays under 100. For larger programs, more memory is automatically obtained and used.

The Syntax Analyzer. The Syntax Analyzer is built to operate on one I-D pair at a time. It is this characteristic that allows the compiler to never be more than one line of text behind the user input. Part of a statement may already be "compiled" while another part may not yet have been input.

A control routine selects which part of the analyzer is to be called. The selection is made entirely on the basis of the delimiter in the I-D pair. Each delimiter has a small piece of analyzer code that knows what to do with the delimiter and its paired identifier.

The delimiter routines break down the statements in reverse Polish form. This type of scan has the property that it can be completed in one left-to-right pass. An I-D pair is, therefore, looked at only once,

another characteristic which is necessary for continuous compiling.

A reverse Polish string has the form:

<operand, operand, operator>

where an operand is either an identifier or another Polish string.

Example:

The Expression: A + B
 In reverse Polish is: A, B, +
 The Expression: A + B * C
 In reverse Polish is: A, B, C, *, +

The order in which the operators appear in the string is governed by the assigned precedence of the operators. All delimiters have precedences and enter into the Polish string. Actually the Pseudo Machine Code corresponding to the delimiters is inserted into

the string and is basically in the form necessary for execution.

A push down stack is used to queue up delimiters appearing in a statement until a delimiter of low enough precedence is found that will permit the delimiters in the stack to be inserted into the string. This process is repeated until the end of the statement is reached.

A simple example is offered of the basic steps performed during translation:

The Expression: A := B + C * D;

1. is first broken down into I-D pairs.

IDENTIFIER	DELIMITER
A	:=
B	+
C	*
D	;

I-D PAIR (PRECEDENCE)	STACK	CODE	PERFORM
a) A := (1)	empty	empty	1) Generate: "A" 2) Stack := (1)
b) B + (2)	:= (1)	A	1) Generate "B" 2) Stack + (2)
c) C * (3)	:= (1) + (2)	A B	1) Generate "C" 2) Stack * (3)
d) D ; (0)	:= (1) + (2) * (3)	A B C	1) Generate "D" 2) Unstack * (3) 3) Generate "*"
e) D ; (0)**	:= (1) + (2)	A B C *	1) Unstack + (2) 2) Generate "+"
f) D ; (0)	:= (1)	A B C * +	1) Unstack := (1) 2) Generate ": ="
g) D ; (0)	empty	A B C * + :=	1) Generate;

Resultant Code: A,B,C,*, :=, ;

** For steps e through g, the same delimiter (;) is being processed. This is because unstacking action is taking place.

2. Precedences of delimiters are found during the table lookup phase.

DELIMITER	PRECEDENCE
: =	1
+	2
*	3
;	0

3. I-D pairs are processed one at a time. The stacking algorithm is basically as follows:

If the precedence of the current delimiter being examined is less than or equal to the precedence of the entry at the top of the stack (=0 if stack is empty), then take the entry off the stack and generate the proper PMC for it. If the above condition is not true, then put the current delimiter with its precedence onto the top of the stack.

4. At this point, an element is formed and a compiled statement emerges. The resulting code that has been depicted here is but a symbolic representation of the PMC that are actually generated.

The scanner and analyzer portions of the compiler are built to operate in parallel in a time-sharing system. The scanner must issue text writes to secondary storage and is, therefore, swapped from core periodically. Previous to write, the text to be written has been broken down into I-D pairs which

have been loaded into a ring buffer. Hence, the analyzer may be started up as a separate program to process the pairs and, therefore, complete the translation process. The analyzer is dismissed from operation when its job is done.

The core layout for this operation looks typically like the diagram in Fig. 3. Core is depicted in pages of 2048 words. A typical configuration might have, say, three pages of PMC.

The text dictionary is a table of pointers to symbolic text on secondary storage. Since all code is re-entrant ALGOL never takes more than three pages for any number of users on the system. An additional two pages unique to each user are required for table storage.

CCS FORTRAN IV

The FORTRAN Language specified by the Subcommittee of the American Standards Association, Sectional Committee X3,⁷ is a subset of CCS FORTRAN IV.

Compiler Description

The compiler incrementally translates each source statement into an element consisting of pseudo machine code and program structuring parameters. Modifications to the source program can be made at the statement level without recompiling the entire program.

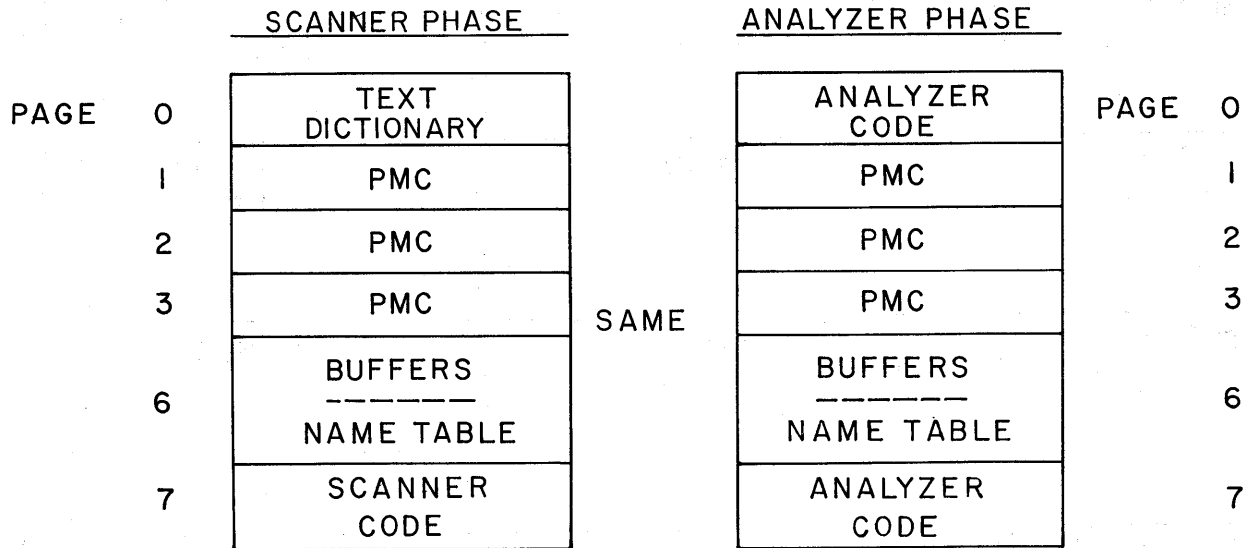


Figure 3. Core layouts during compile.

Text Input. The Input/Edit consists of a collection of primary subroutines that perform the following functions:

GET transfers the current character to the higher level program making the request. Successive GET's obtain the same character.

ADV transfers the next character to the higher level program and advances its character pointer.

MRK places a pointer to the current character in a table.

RST resets the current character pointer according to the last entry in the table and removes the last entry from the table. Execution of more RST's than MRK's constitutes an error.

CLR clears all entries from the table.

These subroutines are actually programmed operators or POPS that are a unique hardware feature of the SDS computers. The GET and ADV requests initially cause an entire line to be input to a text buffer and the required editing performed. When the current character pointer advances to the last input character, a new line is read in. Lines that terminate with a carriage return cause the entire mechanism to be re-initialized when the calling programs have advanced the current character pointer to the carriage return character.

Syntax Analysis. The Syntax Analyzer scans the first alphanumeric string in a statement image. If a special quote such as COMMON, DIMENSION, etc., is recognized the corresponding statement type is assumed and the appropriate syntax subroutine is called. Failure to identify a special quote will cause transfer to the arithmetic statement subroutine which resets the scan to the beginning of the statement and performs arithmetic replacement statement analysis. Subsequent failures cause a diagnostic element to be generated.

The Syntax Analyzer makes one left-to-right scan to generate Pseudo Machine Code. The technique is similar to that described for ALGOL in that the analyzer operates on identifier delimiter pairs and that statements are broken down in reverse Polish form. Operators and delimiters are assigned precedence values and a push down stack is created. During the stacking, delimiters or operators of lower precedence cause previous delimiters and operators

in the stack to be unstacked and used for the generation of Pseudo Machine Code.

Uncertainties that exist for a FORTRAN IV statement taken out of context, such as implicit versus explicit declaration of variables, are resolved by the structuring routine prior to the execution of the program.

Conventional FORTRAN compilers often resolve syntax ambiguities created by statements considered out of context by requiring some types of statements to precede others. An example is the statement function vs. subscripted variable name, resolved by requiring dimension statements and statement function definitions to precede the use of the corresponding identifiers.

However, this technique is not appropriate for incremental compilation. Therefore the convention of brackets “[]” enclosing parameters of function calls is adopted, thus, enabling the compiler to produce the correct object code without reference to other parts of the program. This removes odious precedence relationships and creates a more understandable syntax.

Elements generated for storage allocation statements are placed by the structuring routine at the beginning of the program or subprogram in which they appear. At execution time, a definition search routine establishes pointers to variable storage according to the rules governing COMMON and EQUIVALENCE interaction. The definition search routine establishes a unique data reference for each of several uses which might be defined for an identifier within program and subprogram relationships.

Because CCS permits the interfacing of assembly language written programs, it is possible to use existing programs⁸ for intrinsic and basic external functions, conversion and editing of variable width formatted input/output and memory to memory data conversion.

The input to the compiler consists of free form FORTRAN IV source text images from a typewriter console or the system file storage. A semi-colon or carriage return separates statements. A line-feed continues a statement to the next line.

Identifier Mechanics. A name table entry is created by the compiler for each identifier encountered. The associated table entry number is used for structural references to the identifier.

In conjunction with the name table a bit table is maintained which enables the definition search routine to resolve implicit definitions. Each bit of the table corresponds to an associated name table entry and is set depending on the first letter of an identifier name. Explicit definitions will override implicit definitions during the structuring phase, however, the bit table is preserved to provide for implicit definition should the explicit definition be deleted.

Diagnostics. When the compiler encounters a syntactical error during a statement scan, a diagnostic element is generated which describes the nature of the error and locates the symbolic text for the erroneous statement.

Extended Features of CCS FORTRAN IV

Many powerful extensions have been included in CCS FORTRAN IV. These extensions are a natural consequence of the structural environment created by the design of CCS in response to the multiprogramming demands of a time-sharing system. Some of these extensions are listed below.

Statements may be written in free format and are not affected by card boundaries. The semicolon or carriage return is used to separate statements.

Alphanumeric data handling capability has been extended with the inclusion of string variables and string arrays. These declarators, in conjunction with the string manipulating functions included in the system library, provide a powerful string processing capability. This capability encompasses extraction and concatenation of strings as well as the obvious uses of relational operators and input/output of "messages," etc. As mentioned previously, string array elements and string variables are of arbitrary length, removing all considerations of machine dependency with respect to the treatment of alphanumeric data.

Arithmetic expressions may be used in all cases where a single variable is allowed, including subscripts. All expressions may be of mixed modes.

There are no limits to the number of subscripts which may be declared for a dimensioned variable and both upper and lower bounds may be defined via variables, expressions or constants.

Internal subprograms are provided which may refer to variables defined in the parent program without the use of argument lists or COMMON.

Storage allocation may be regulated so that storage is assigned only when required by the execut-

ing program. Programs debugged under other compilers will result in the DIMENSION statement being executed once at the beginning of the program or subprogram in which it appears. However, a dynamic dimension statement of the following form may appear anywhere in the program

$$\text{DYMENSION } v(e_1, :e_2, e_3 : e_4)$$

where e_1 , e_2 , e_3 and e_4 are expressions for evaluating the general bounds of a two dimension array. Note that storage no longer required may be effectively erased by executing DYMENSION $v(o,o)$.

Using features provided by the Pseudo Machine Code, CCS FORTRAN IV includes matrix logic features similar to PL/I. For example, in the statement,

$$A = B + C$$

if A and B are arrays with corresponding subscript bounds and C is a variable, each element in A is set to the corresponding value in B added to the value C.

In addition, the programmer may specify "array structures" which are components of arrays. For example, the statement,

$$A(2,*) = B(2,*)$$

causes each element in A with value 2 in the first subscript position to be set to the corresponding value in the array B.

The arithmetic operators +, -, *, **, / are permissible in matrix operations with the obvious meanings.

Dynamic Data Storage. As has been indicated, CCS FORTRAN IV includes extensions to the syntax for using dynamic data storage. The following coding example illustrates some of the capabilities provided with dynamic storage:

```
DATA K = DATA AREA SIZE
DIMENSION X(DATA SIZE 1), Y
(DATA SIZE 2)
—
DO 100 I = 1, F
100 ACCEPT [X(I)]
—
DO 200 I = 1, S
200 ACCEPT [Y(I)]
—
SCRATCH PAD = K - (F+S)
—
```

```
IF (SCRATCH PAD .GT. 0) DYMEN-
SION Z (SCRATCH PAD)
```

```
—
DYMENSION Z(0), S(SCRATCH PAD/
2), T(SCRATCH PAD/2)
```

```
—
FUNCTION DATA AREA SIZE
DISPLAY ['ENTER SIZE OF DATA
AREA', $]
DATA AREA SIZE = INPUT
```

```
—
FUNCTION DATA SIZE 1
DISPLAY ['ENTER SIZE OF 1ST
DATA GROUP', $]
DATA SIZE 1 = F = INPUT
```

```
—
FUNCTION DATA SIZE 2
DISPLAY ['ENTER SIZE OF 2ND
DATA GROUP', $]
DATA SIZE 2 = S = INPUT
```

```
—
FUNCTION INPUT
ACCEPT [INPUT]
```

```
—
END
```

CONCLUSIONS

It is our hope that the Conversational Compiler System will prove to be a meaningful contribution towards the achievement of effective man-machine interaction.

CCS takes advantage of the concept of time-sharing to permit a rapid dialogue between the problem solver and the computer in terms of the language used to describe the problem or process. This is accomplished by raising the debugging facilities to the level at which the problem was defined.

Furthermore, since the programming language is itself critical to the definition and solution of a

problem, CCS has been designed to facilitate the implementation of additional languages and, hopefully, to allow each language implemented to take a form which is much more general and flexible than its existing counterpart, if any. In fact, the inclusion of features in a language on the system found in any other language on the system, merely involves the design of a suitable syntax to express that feature.

In final analysis, only extensive testing and accumulation of experience will show whether these efforts will help close the communication gap between man and machine.

REFERENCES

1. M. Pirtle, "Modifications to the SDS 930 Computer for the Implementation of Time Sharing," Document No. 30.10.10, Department of Defense Contract SD-185, U.S. Printing Office, 1966.
2. K. Lock, "Structuring Programs for Multiprogram Time-Sharing On-Line Applications," California Institute of Technology, Pasadena, Calif.
3. B. Randell, and L. J. Russell, *ALGOL 60 Implementation*, Academic Press, London, 1964.
4. B. W. Lampson, "Time-Sharing System Reference Manual," Document No. 30.10.30, Department of Defense Contract SD-185, U.S. Printing Office, 1966.
5. C. S. Carr, "Question Answering System," Document No. 30.60.40, Department of Defense Contract SD-185, U.S. Printing Office, 1966.
6. P. Naur, ed. (with amendments by Woodger, M.) "Revised Report on the Algorithmic Language ALGOL 60," International Federation of Information Processing, 1962. (Also in Communications of A.C.M., Vol. 6, No. 1, pp. 1-17, 1963.)
7. Specifications for FORTRAN established by the Subcommittee of the American Standards Association Sectional Committee X3, as reported in the Communications of A.C.M., vol. 7, no. 10, pp. 590-625, 1966.
8. Scientific Data System, SDS FORTRAN IV Reference Manual 90 11 07A, January, 1966.

PERFORMANCE OF A MONITOR FOR A REAL-TIME CONTROL SYSTEM

Erna S. Hoover

Bell Telephone Laboratories, Holmdel, New Jersey

and

Barry J. Eckhart

*Bell Telephone Company of Canada, Montreal, Canada**

INTRODUCTION

In planning the program design for a real-time control system, it is essential that the monitor program use an appropriate policy for scheduling work. The performance of the system will depend upon a number of things; but a wise choice of scheduling algorithm is essential.

If the system to be controlled is complicated enough, the advantages and disadvantages of particular scheduling methods become apparent only after the performance of the system has been extensively studied. Frequently it is not possible in early stages of program design to predict the behavior of a system satisfactorily; in such cases simulation is a help to understanding, which in turn leads to informed choices in the design and manner of deployment of the system.

The use of simulation is well suited to the study of large real-time control systems and of computers operated in a multiprogramming mode. Such systems usually have the following properties:

- Demands for control action arrive in a random manner.
- Different kinds of demands have a different tolerance for system delays.
- These tolerances must be met under all load conditions.

In order to use the system efficiently, the scheduling routine itself should use a minimum of system time and arrange the order of processing for the rest of the work so that the system employs its time in an efficient manner. This is especially important when the system is operating under heavy demand.

A study of such a system has been made. By relying heavily on simulation, the suitability of a particular scheduling algorithm has been determined. The method chosen for scheduling machine time affects the delay to jobs of different urgency by checking relatively more frequently to see whether jobs of higher urgency are waiting. Unlike the more familiar priority allocator, the relative frequency allocator serves jobs of a lower urgency when their turn comes, no matter what more urgent job is waiting. As a result, even in very busy periods, the ratio of delays experienced by jobs of two given urgencies

* Mr. Eckhart contributed to this work during an assignment at Bell Telephone Laboratories, Holmdel, N.J.

is likely to change less than would be the case under a priority system. No job is put off indefinitely.

During periods of very heavy demand for machine time, this kind of monitor exhibits a "snowballing" effect; once encountering an unusual amount of work, it goes through its fixed order of service at an unusually slow rate, processing all the work it finds. Since work is likely to be waiting for it on the next cycle, one slow cycle is likely to lead to another. Most of the delays to jobs occur in these slow periods. By means of simulation, it was discovered that the same volume of work can be served with less delay if the work is organized in a suitable way. The fixed order of serving work must be arranged so that the monitor is not likely to try to serve extremely large amounts of work of a given urgency before going on to the next class of work. Such an arrangement was found, and it was shown by simulation to improve the grade of service to various jobs.

DESCRIPTION OF THE SYSTEM TO BE SIMULATED

In certain communities, Bell System customers are currently served by a new kind of automatic switching system: A stored program of over 100,000 words controls the telephone equipment and responds to the customers' demands for service. This electronic switching system must serve thousands of telephone customers in a given exchange area and respond to their demands for service without undue delay.¹ The system has a monitor program which allocates the processing time of the machine among the many tasks which an automatic switching system must perform. Like many real-time control systems, the basic requests for service arrive at random and are not under the control of the system. The system can be considered as a single server processing multiple queues where the service discipline is given by the monitor.

Types of Tasks

The stored program control of the Electronic Switching System, or ESS, spends the major portion of its time processing telephone calls.² There are, however, other tasks which must be done. In order to permit the telephone companies to determine the volume of calling and to provide sufficient equipment for it, the program measures the amount of traffic and periodically reports the results for these administrative purposes.

The program also makes routine tests of the equipment to check that all is working well.³ Whenever a fault is found in a particular piece of equipment, it is immediately switched out of service. The program then diagnoses the fault and prints out the results of the investigation so that a repairman can readily replace the portion of the equipment which has gone bad. The monitor program must assign sufficient machine time at appropriate times so that all these tasks, call processing, administration, and maintenance, are properly performed.

The problems of real-time control encountered in this system are similar but not identical to those involved when a multiprogramming monitor operates a computation center. In both cases a model of the manner in which the processing jobs are done and of the monitor scheme can be constructed. Fortunately, in the case of the telephone system, much is known about the nature of telephone traffic so that the demands made upon the system can be realistically simulated, whereas much less is currently known about the demands placed on a multiprogrammed computation center.

System Delays

If the monitor system allows excessive delays the following two general types of penalties can occur:

1. Equipment will be inefficiently used. In a telephone switching machine there are many groups of equipment items supplied in quantities based on the expected traffic. If the system organization causes excessive delays between the occurrence of an event which makes available one of these equipment items and the actual releasing of the equipment item by the system, the system monitor is in effect keeping that equipment item busy. This applies to both hardware equipment items and temporary memory registers. This artificial holding of items within a group results in decreasing the capacity of the group to carry traffic.
2. Service delays are increased. The sum of the delays imposed by the monitor form a component of the service delays experienced by the customer. In this way the system organization directly affects the service given the customer.

Fortunately jobs vary enormously in the amount of delay they can tolerate. Delays on the order of tens of milliseconds, which seem long when measured on the microsecond scale of machine instructions, will barely be noticed by telephone customers. Permissible delays for different types of work vary from tens of milliseconds to a second or so. The program also deals with input/output signals where the system must sample the state of certain circuits every 11.5 milliseconds or risk the loss of accurate information. The monitor takes advantage of this difference in the tolerance of delay for different jobs when it organizes its work.

Statistical Nature of the Major Demands on the System

The system is arranged so that it can devote the bulk of its time during the busiest hours of the day to processing traffic. Even at these times, a certain minimum, on the order of 1 or 2% of system time, must be spent in routine checks of the entire system, including the data stored in memory, the circuits of the memory and processor, and the circuits controlled by the system. Occasionally the equipment will malfunction. This requires the attention of the diagnostic programs which may wholly occupy the system for a brief time. Since the hardware is designed to provide reliable telephone service day in and day out, the occurrence of faults is very rare; consequently a negligible amount of the total time of the system is spent in diagnosing actual troubles.

The work of the machine in processing a telephone call is broken into a series of discrete tasks, ranging in number from 2 to over 20, depending on the complexity of the call. For example, when a customer dials, the machine spends a small fraction of a millisecond to see whether dialing is complete. The call can proceed either by ringing the desired party, returning busy tone, or by routing the call to another central office. These tasks are more complicated and typically may take from 1 to 20 milliseconds to complete. Thus some tasks place a demand on the system which is 20 times or more the demand of other tasks.

Not only does demand for call processing time vary with the type of input, but the number of inputs also varies. Studies made over the years on the behavior of telephone customers indicate that the number of requests for service arriving in a given small period of time can be expressed as a Poisson

distribution. Similarly, conversation times are exponentially distributed.

In order to accord each type of job the service appropriate to it, jobs are divided into two groups; those which have a tolerance for delay of less than .2 of a second and those which have more than .2 of a second. Many of the jobs in the first group must be performed within a tolerance of 5 milliseconds, some within a few microseconds. These jobs are subdivided accordingly.

Many of the jobs in the second group can tolerate delays of over a second in a small fraction of cases. However it is desirable that the *average* delay should be less than .2 of a second for most types of jobs and considerably less for some. Accordingly, these jobs are also subdivided into types according to the relative amount of delay which was judged tolerable.

Meeting Small Timing Tolerances

When the system was designed, two methods for processing the more urgent jobs were known to the designers. One method would require that all processing sequences be broken into very short runs, on the order of 100 microseconds. Between each run, the monitor would check to see if any of the urgent jobs were waiting.

The other method would provide an interrupt circuit which would interrupt the program in control of the machine and cause a transfer to the urgent job.⁴ Data pertaining to the interrupted program would be transferred from the flip-flop registers of the machine, preserved in memory, and restored when the interruption is over. Therefore the interrupted program would not be aware the interruption had occurred.

The second method was chosen for several reasons. To require that programmers break their programs into such short sequences is at best burdensome and at worst impossible. Large control programs perform make extensive use of subroutines. If the subroutines themselves transfer to other subroutines conditionally, the program which calls the first subroutine cannot control the running time. Hence, either severe requirements must be placed on the structure of subroutines, or such a restriction cannot be met at all. But to meet such severe requirements would be burdensome for the programmers. Also, such an arrangement would undoubtedly result in the machine's spending much of its time in overhead. Furthermore, on the rare occasion when a

malfunction occurs, it is desirable to transfer to the maintenance programs as soon as possible. The interrupt method permits the transfer within a few tens of microseconds.

Figure 1⁵ illustrates the time-sharing between the jobs performed during an interruption and the base level jobs. During normal operating conditions the urgent jobs are mainly of an input/output character. To meet their service requirements, the machine is equipped with an interrupt circuit which acts every 5 milliseconds. During each 5-millisecond interruption a number of these urgent jobs are scheduled in the order from least to greatest tolerance for delays. First in the list is the most sensitive job which has a tolerance of 1.5 milliseconds. Obviously these programs must be short.

Very rarely, a hardware fault will occur. The interrupt circuit will then break in and pass control of the machine to a program that will pinpoint the defective unit and switch it out of service. A hierarchy of interrupt levels for maintenance exist so that the occurrence of more serious faults may interrupt the input/output tasks and the programs which analyze less serious faults.

When a higher level interrupt program is finished, control is passed to the program which was interrupted. Control then continues down the levels of interruptions. At each level the data which the interrupted program had held in the flip-flop registers

of the machine are replaced before control of the machine is returned to that program. The maintenance programs that run during an interrupt are also designed to run only as long as necessary. Once the faulty unit is recognized and is switched out of service, the program stores its findings for another program which will operate in the base level to find the exact cause of the fault.

The Hopper-Queue System

In order to be able to serve the different types of work in the base level appropriately, the various types of work must be identified. Usually these base level programs are initiated by an input/output program which runs during an interruption. Fortunately, a given input/output program usually initiates jobs which can tolerate a uniform amount of delay. Therefore a given input/output program will file all its work in a specified area of memory called a "hopper". The monitor then has the problem of serving the hoppers, each of which contain work with a certain service requirement.

At several stages in the processing of a given call, the program will require the use of an item of memory or equipment from some group of items. There is a small probability that such an item will not be immediately available. For example, each installation is supplied with a generous set of ringing circuits. On the rare occasions when a call requires a ringing circuit and none is available, that call is placed in a queue to wait for a circuit to become free. The monitor thus serves a set of queues for items of equipment and a set of hoppers for work recently found by the programs that operate on the interrupt level.

Choosing the Algorithm for Serving the Hoppers and Queues

In choosing the monitor algorithm the designers of the system were chiefly concerned with the nature of the distribution of delay that each type of job should experience. The precise form of the distribution of delay for each type of job and the exact value of the average delay which is tolerable could not be specified at that early stage in planning the system. Usually the average delays to serving jobs affected the use of equipment and hence affected economic tradeoffs. The exact values of these tradeoffs were not known at that early stage in planning. It was easier to specify the limits on the tails of the delay distributions because extreme delays interfere with

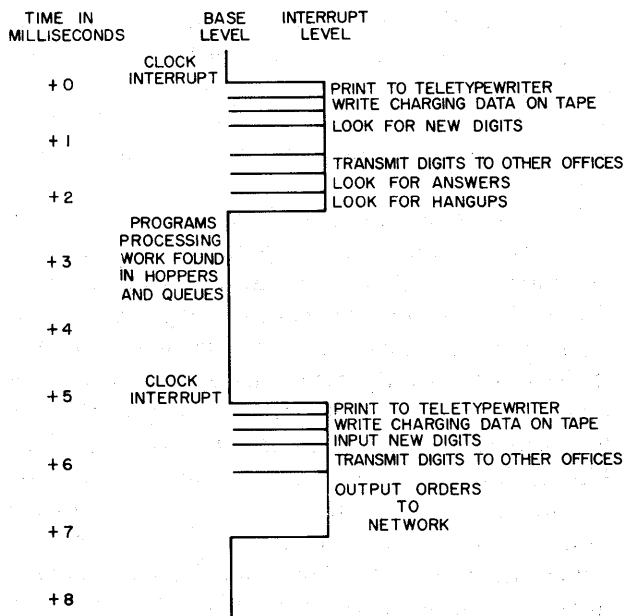


Figure 1. Time sharing between base level and input/output programs.

the proper processing of calls. The designers were able to form a rough judgment about the ratios between the average delays which each type of job could tolerate. Accordingly, they assigned each type of job to one of five classes, designated "A", "B", "C", "D", and "E", according to the delay which would be tolerable. They felt that during typical busy periods in a large installation the B class of jobs ought to experience about twice the delay of the A class of jobs, the C jobs about twice the delay of the B class of jobs, and so on. E jobs had a somewhat different character because they consist of routine maintenance and administrative jobs, whereas the other classes consist of the call processing jobs. In most cases the worst delay experienced by an individual job of a given type could be four or five times the average delay for a job of that type. As the system became busier, the delays to each type of job could be expected to increase. However, the designers decided that even during the busiest period which an installation might encounter, the ratios between the delays to jobs of different classes ought not to change by more than a factor of 10 from the ratios encountered during a more typical busy period.

Many aspects of a system influence the delay to jobs. The average level of demand for machine time, the statistical variations of the demand, the distribution of the running times of jobs, and the proportion of machine time used by jobs of each class all have an effect on the delay in addition to the effect of the particular monitor algorithm used. Some of these characteristics of the system were known when the monitor was chosen, but their exact effect on delays could not be determined without a simulation.

However, the designers were aware of the general effect of certain of the system characteristics. The machine was likely to experience large fluctuations in demand for machine time, which would tend to increase delays. Most jobs were short, 15 milliseconds or less, which tends to reduce delays. One inevitable portion of the delay is the time spent waiting for the current job, including interruptions, to finish running. The other portion, of course, consists of the time spent in serving other jobs before the particular job is served. This will depend on the choice of the monitor algorithm and on the amount of total machine time required for each class of jobs. The latter fact could not be known when the monitor was chosen.

In the light of the considerations known to them, the designers chose the algorithm illustrated in Fig. 2 for serving the bulk of the base level work. The

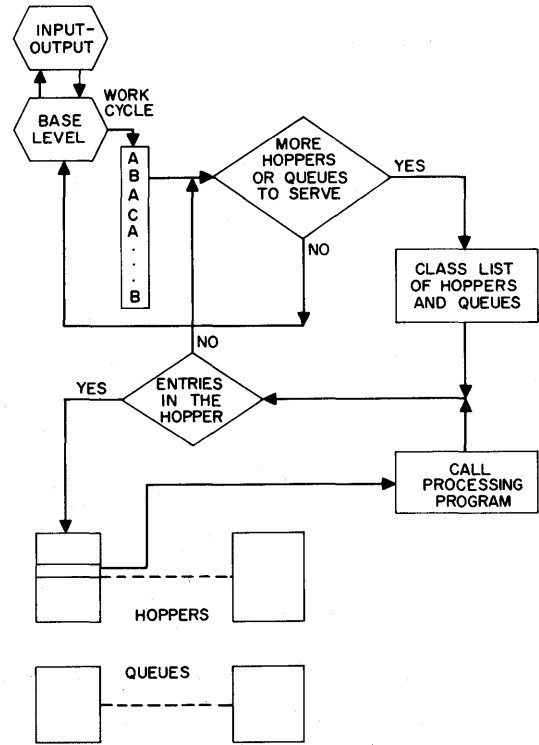


Figure 2. Organization of base level work.

monitor goes to each hopper in turn, looks to see if there is work waiting, takes each waiting job from the hopper in the order of first come, first served, and turns control of the machine over to the appropriate processing program. This program may be interrupted for input/output work, but control is returned to it, and it is allowed to finish before any other base level program is started. When the job in a hopper is finished, the next job is taken. This continues until the hopper is empty. Then the monitor goes on to the next hopper or queue in the particular class being served. Entries are taken from queues only if there is an appropriate equipment item available. When the monitor has served all the hoppers and queues in a given class it then goes on to the next class indicated by the work cycle. The monitor cycles through the five classes in the manner shown in Fig. 3.

This type of monitor gives different grades of service to the different classes of jobs by serving the more urgent jobs relatively more often than the less urgent ones. If there is no call processing work to do at all, the monitor cycles through all the hoppers and queues of Classes A through D looking for work and finding none. Since the work in E is routinely scheduled, the monitor will allocate a large portion of the entire machine time to routine maintenance.

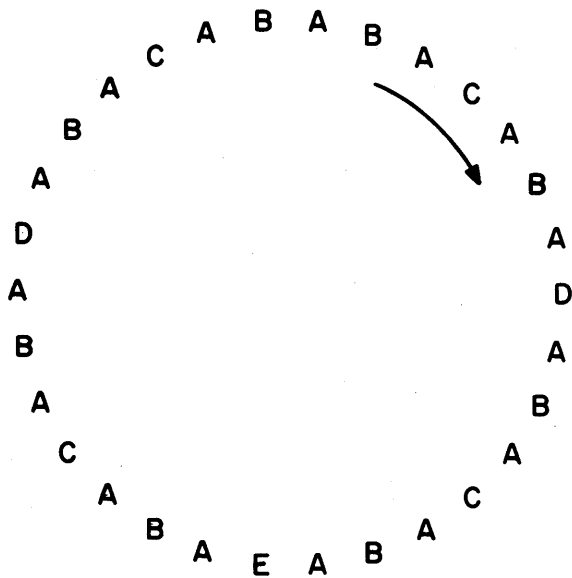


Figure 3. Monitor work cycle.

When the call processing work builds up, the time it takes to come full cycle lengthens; but the machine devotes less of its time percentagewise to Class E work when demand for machine time is great. It is desirable to reduce the proportion of time spent on routine matters when call processing demand is great, but to perform some minimum amount even when the machine is very busy.

A monitor of this kind can be characterized as a relative frequency allocator of system time in contrast to the more familiar priority allocator. A relative frequency allocator obtains different grades of service for different types of work by giving the more urgent work relatively more chances to be served. However, when the place in the frequency table is reached where a class of less urgent jobs are scheduled, such a relative frequency allocator does not check to see whether urgent jobs are waiting. It processes the less urgent class of jobs. It is for this reason that even low priority jobs receive a minimum grade of service. No job is delayed indefinitely.

A priority allocator was also considered. When a priority allocator is used, the queues and hoppers which hold jobs of *all* the higher priorities are checked before *any* job of the next lower priority is begun. Under such an arrangement, when the machine gets very busy, the ratio of the delays of the lower priorities to the higher lengthen appreciably. Sometimes the lower priority jobs are put off indefinitely while the machine occupies itself with the higher priority tasks.

A strong form of the priority method is used in ESS in the levels of interrupt jobs. In addition, a weaker form is used for a small fraction of the base level jobs. A few of these jobs are required to wait only as long as the machine takes to finish its current base level jobs including interruptions. The machine checks to see if such a priority job is waiting whenever a current job finishes. If so, it turns control over to the priority job immediately. Fortunately, these base level priority jobs constitute a small fraction of machine time. If they occupied a large portion of machine time, the delay they experience would consist to a significant degree in time spent waiting for another priority job to finish. Thus, both the priority method and the relative frequency methods are used in ESS, but the bulk of the time used by base level work is allocated by a relative frequency method.

Because exact delay requirements were not available when the monitor was programmed, the assignment of jobs to frequency classes and the composition of the frequency table were arranged so that a simple change in the data consulted by the program can alter the grade of service to the various hoppers and queues. The monitor program which was written has the additional advantage that the check for the priority jobs and the check for the next job in the schedule together consume relatively little overhead time in the machine.

DESCRIPTION OF THE SIMULATION PROGRAM

In order to obtain the delay distributions for the various kinds of jobs, it was necessary to simulate the behavior of the machine. This simulation uses a fairly detailed model of the operating system itself and an almost exact model of the monitor.

The behavior of telephone customers is simulated according to the known characteristics of telephone traffic. Calls arrive according to a Poisson distribution; the lengths of conversations are distributed exponentially. The distribution of times which users take to dial and also to answer a ringing telephone are closely approximated.

As is usual in studying complicated systems, a simplified model of the system was used to reduce the burden of programming the simulation. Judgment must be used in choosing the features to be eliminated lest important characteristics of the true system be omitted from the model. In the case of the ESS, most of the program sequences are entered

very infrequently and do not make a large demand on system time. Hence, it is possible to omit these aspects of the real system from the simulation without affecting the ability of the model to accurately reflect the performance of the real system.

Six main types of simplification were made:

1. Since maintenance interruptions which report errors are very infrequent, only the interruptions which represent the input/output tasks are performed in simulating the processing of telephone calls.

2. Only the most common types of telephone calls, which in fact make the predominant demand on system time, were simulated.

3. Routine maintenance and administrative tasks, which are performed during frequency Class E, are simulated at first as if they always took a constant amount of time to run. Later runs were made which included the variation in running time. The most significant difference between the routine maintenance and call processing programs lies in the fact that the machine will always encounter at least 10 milliseconds of work when it enters the E class. It may find no work at all when it enters frequency classes containing call processing jobs.

4. The fault recognition and diagnostic programs which are triggered when a hardware trouble occurs were not simulated for two reasons. They occur so infrequently that they are not a typical hourly experience for the machine. When they do occur, their effect is analogous to effects already simulated by the interruptions caused by input/output and the base level call processing programs.

5. The interrupt level of priority jobs were not simulated in exact detail. In a large installation, it is usual for the interruption triggered by the 5-millisecond clock to take, on the average, about 50% of the 5 milliseconds. But the actual times will vary depending on the work which the input/output programs find to do. These interruptions have the effect of stretching out the time to do call processing work by a variable amount. The variation in interruption times and its effect on the call processing delays were simulated. However, in the real system, longer than average interruptions are caused when unusual amounts of work for the base level programs were found. Although the simulation imitates the variations in finding jobs for the base level work, it uses separate programs to generate the length of the 5-millisecond interruption and the new inputs to the hoppers. Hence the occurrence of long interruptions

is not correlated in the simulation with unusually large amounts of work found for the hoppers. Since the 5-millisecond interruptions generally varied from 1 to 4 milliseconds and since most delays will include at least five such intervals and usually many more, it was judged that the simplified manner in which the interruption intervals and the work to the hoppers was generated would not distort the results appreciably.

6. Of the priority types of jobs which are performed in the base level on a priority above the frequency classes, only one type was simulated. When an input program can find no space in the memory allotted for its hopper, it puts in a request to the monitor to empty that hopper as soon as it finishes the base level job it is doing. It was judged to be desirable to provide ample space in hoppers, and consequently this situation in fact was never met. The other priority jobs were ignored because they constitute a small fraction of total demand on the time of the machine.

The simulation program generates the traffic to be offered to the system and then simulates the performance of the system as it processes the traffic. Finally, as output, it prints a number of tables showing the distribution of times taken by visits to each hopper, the distribution of times spent in processing the work of each hopper, the delays encountered by the various types of jobs, and the overall delays accumulated by each call. The time taken to complete each cycle of the frequency classes is printed in sequence as the simulation runs.

The program will realistically simulate installations which vary widely in equipment configuration and also in traffic patterns. By changing data cards, the description of the equipment and traffic to be simulated can easily be changed. Thus one can simulate conditions not yet encountered in actual operation.

The simulation was coded in FAP, the assembly language of the IBM 7094 computer. Assembly language was chosen because it permits more efficient packing of data in memory. Both the program and large amounts of data must be kept in the core memory of the IBM machine at the same time. Hence a high packing efficiency was needed to insure that the 32,000 words of 7094 core memory would not be exceeded. Even though subroutines were used wherever possible, the program consists of 11,300 words. Of these, 20% constitute the representation of the logic of call processing sequences of ESS.

Flow charts were drawn of those ESS program sequences which constitute the bulk of the demand for machine time. Blocks on these flow charts were represented in the simulation program by subroutines which simulated the appropriate action. The flow charts themselves are represented by a series of calling sequences to these subroutines. Although much simplified, the remaining model is still a complicated logical structure.

An important element of realism in the simulation is provided by the method for simulating the number of instruction cycles which the ESS machine requires to do each job. These cycles are entered as data to the simulation. Before the ESS programs were finished, estimates were made of the cycles which the program used for each one of its tasks. As the actual program became available, measurements were made on the machine itself, and the appropriate program sequences were examined in detail, so that accurate cycle counts were used in the simulation. In spite of efforts to simplify the tasks, the formulation of the model of the ESS logic, the determination of realistic cycle counts, and the design and writing of the simulation program all required considerable effort.

PROCESSING DEMAND AND MONITOR INTERACTION

Certain characteristics of system behavior can be predicted from a consideration of the monitor algorithm itself; other characteristics require simulation. First let us consider what can be concluded by considering the monitor algorithm.

Analysis of Monitor Algorithm

Processor Occupancy. The tasks which comprise input/output functions and call processing in the base level represent the real-time demand on the system. The other types of tasks, routine maintenance and administration, together with the time taken to cycle through the hoppers and queues, are not dependent on the real time but are a function of the way the monitor schedules its work. These latter tasks, although necessary, constitute overhead. Therefore, the occupancy of the processor is defined as the fraction of time it spends on input/output and base level call processing. If there is no call processing to do the system simply performs more maintenance and spends a higher percentage of its time looking through hoppers that are empty. Together, the time

spent in overhead tasks plus the time spent in processing real-time demands account for all of system time.

Effect of Processor Occupancy on Work Cycle Time. As more telephone calls are processed, the occupancy of the system increases and there will be more input/output work. Thus interruptions also take more total time. As the monitor cycle goes through the hoppers and queues it will find work and transfer control of the processor to the appropriate processing programs. Since the processor now takes time to process the work, the total time taken to get through the cycle of work increases. The length of a complete cycle, including routine maintenance work in Class E, will be called the Class E revisit time.

- Let V = average E revisit time,
 K = a constant time to cycle through the hoppers and queues including routine maintenance, and
 x = machine occupancy, i.e., call processing and input/output.

Then $V = K + xV$, or

$$V = \frac{K}{1 - x}$$

The curves shown in Fig. 4 show the length of time to revisit E as a function of occupancy for several different amounts of system overhead. In all the curves shown, the time to go through the hoppers and queues looking for work was held constant. The period of time spent in Class E doing routine maintenance and administration was chosen as 30, 10, and 1 milliseconds respectively, on the three curves

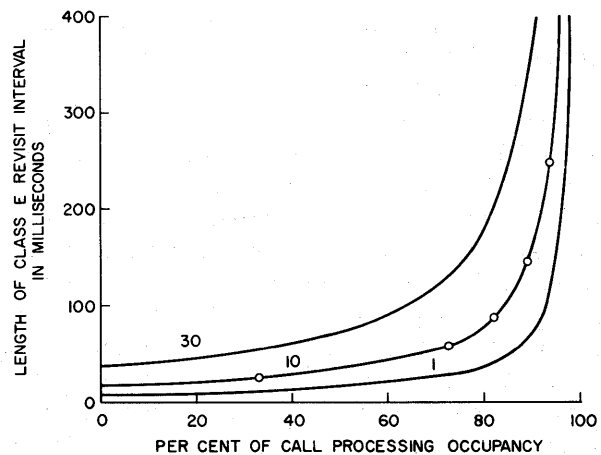


Figure 4. Length of Class E revisit interval vs system occupancy for various Class E constants.

presented. The middle curve shows the theoretical plot, together with points derived from running the simulation. The other two curves show theoretical plots only.

The above expression is, of course, for the average Class E revisit time. The average revisit time for any other class can also be derived. For any class, the revisit time is defined as the time from the start of processing that class to the next start on the same class.

Given a particular order for cycling through the classes of work, let:

M_y = the number of times Class y appears in the list,

V = the time to cycle through the list once, and

V_y = the average time between visits to Class y .

Then

$$V_y = \frac{V}{M_y}$$

As a result, in the case of the ESS list, the average Class C revisit time is $\frac{1}{4}$ of the Class E revisit time at any given occupancy, since C occurs in the list 4 times, E once.

If the demand in some system is uniformly distributed, the frequency of processing work ought to have a major effect on the delay characteristics of the system. If the demand varies greatly, because the number of inputs per given unit of time vary, because the input programs bunch them together in different ways, and because the amount of machine time demanded by different types of input vary, the time to process each class of work will vary. As a result the time to revisit a given class will also vary. Under these circumstances the average delay to jobs of a given type will be affected by the revisit times when jobs are waiting, which is not under the control of the monitor. One might therefore expect the average delay to jobs of a given class to be about half the average revisit time weighted for the number of jobs found waiting.

Since the input programs bunch work and since some types of urgent jobs take up a great deal of system time, one could not expect these estimates of the average delays to be exact. Furthermore, although one confidently might expect that the distribution of delays to a given class would be no worse than the distribution of revisit intervals when work was found waiting, it was necessary to simulate to obtain both of these distributions.

Table 1. Revisit Behavior of the Monitor for Jobs of Different Classes, Machine at 96.5% Occupancy

Class	Average Revisit Time milliseconds	Ratio of Revisit Time of Class D to Class Shown	
		Predicted	Simulated
A	38	7.5	7.1
B	70	4.	3.9
C	138	2	1.9
D	272	1	1.0
E	544	0.5	0.5

Simulation Study of Monitor Characteristics

Delay and Revisit Data Obtained from Simulation. Tables 1 and 2 show the kinds of data pertaining to delay characteristics.

As shown in Table 1, the average revisit times to various classes fairly closely approximate the times expected from the frequency of occurrence of each class. However, if one omits the times when no work was found waiting, the average revisit intervals for typical jobs in each frequency class weighted according to the number of jobs found waiting are shown in Table 2a. They are substantially higher than the revisit intervals which include the occasions when no jobs are waiting. Table 2a also shows the average delay for the same types of jobs. As expected, they are only very roughly half of the weighted revisit intervals. The ratio of the average delays in each class is not quite the same as the ratio of the revisit intervals shown in Table 1. The standard deviations are large, but acceptable to the designers of the system.

Table 2b shows the same data as shown in Table 2a for a machine which is very busy but not so extremely loaded as the machine for which data is shown in Table 2a. As was to be expected, the average delays and average revisit intervals weighted by the number of jobs found waiting are markedly less. The ratio of average delays to weighted revisit intervals approximates 0.5 more closely.

In this case, although the difference in delays to the different classes is still significant, it is not as pronounced as in the earlier case. The ratio of the delays have changed but not by as large a factor as 10.

Assignment of a job to a particular class has a strong effect upon the delay which it will experience, but other characteristics of the system also affect the

Table 2. Data Pertaining to Delays and Weighted Revisit Times for Jobs of Different Classes

Typical Type of Job of Frequency Class X	(1) Average Weighted Revisit Interval milliseconds	(2) Average Delay milliseconds	Ratio of (2) to (1)	Ratio of Average Delay of Class D to Class X	Standard Deviation of Delay milliseconds
<i>a. Machine Running at 96.5% Occupancy</i>					
A	81	38	.46	6.1	31
B	146	60	.42	3.9	60
C	237	121	.51	1.9	105
D	376	233	.62	1.0	165
<i>b. Machine Running at 88.2% Occupancy</i>					
A	37.9	20.6	.54	3.3	17.3
B	61.5	34.2	.55	2.0	27.8
C	101.5	49.1	.49	1.4	45.4
D	135.9	67.0	.49	1.0	60.5

delays experienced. Because both the average amount of delay and the standard deviations increase markedly as the system becomes very busy, the manner in which the monitor behaved at very high loads was investigated.

Figure 5 shows a distribution of Class E revisit times for an ESS running at an occupancy of well over 90%. The very short visits occur when the system happens to find little to do, the very long ones when a large amount of work happens to come into the hoppers. Most of the system delays, especially those which may exceed tolerances, result from these long visits. Both the effect of monitor constants in Class E and the effects of assignment of call processing jobs to Classes A through D were investigated.

Effect of Monitor Constants on Long Delays. We have seen that the constants which represent over-

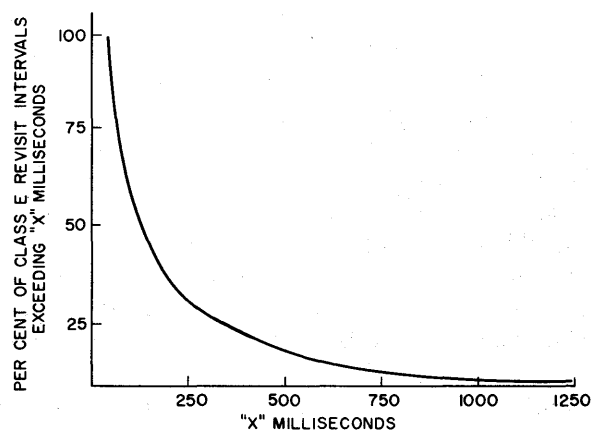


Figure 5. Distribution of Class E revisit intervals when ESS is operating at a high occupancy.

head determine the average Class E revisit time when the system runs at a given occupancy. However, the choice of overhead constants does not strongly influence the occurrence of long visits and long delays, as the following results indicate.

Figure 6 shows results obtained from the simulation giving the distribution of Class E revisit times for three different values for the monitor overhead constants. In each case the system was offered and carried the same traffic load. This load caused the simulated system to run at an occupancy of 93 percent in each case. The three cases represent different arrangements of the overhead constants. The A curve shows the arrangement whereby the lumped constant in Class E is set to 10 milliseconds, whereas the distributed constant required to look through all the

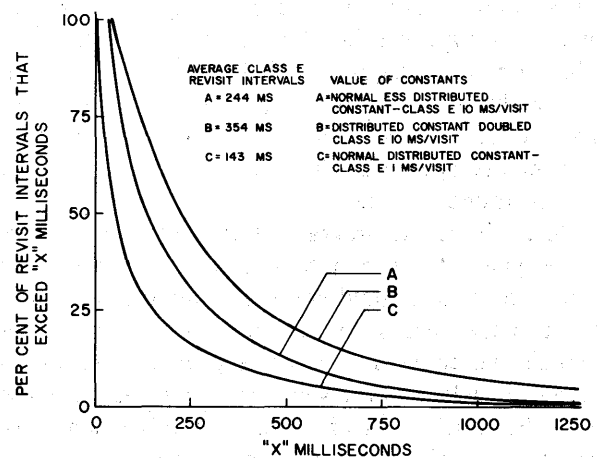


Figure 6. Effect of changes in monitor constants on Class E revisit intervals.

classes of hoppers and queues is somewhat less than 10 milliseconds. The B curve shows the effect of doubling the distributed constant, leaving the lumped constant as in Curve A. The C curve shows the effect of reducing the lumped constant from 10 milliseconds to 1 millisecond, leaving the distributed constant as in Curve A.

As predicted, the average Class E revisit times were proportional to the monitor constant, since the simulated system ran at the same occupancy. However, as can be seen from the curves, the different choices of constant did not have a strong effect on the percentage of long Class E revisit times.

Figure 7 shows, for the same length of run, the actual number of long visits for each of these cases.

For comparison, we show in Fig. 8 a distribution of the delays to the work performed in Class A. It will be seen that the differences between the three versions of the monitor are slight indeed.

We therefore conclude that long revisit times and long delays are primarily a result of the way the monitor serves the highly variable demand for call processing.

Effects of Variable Demand. a) Correlation of Lengths of Class E Revisits. As mentioned earlier, inputs to the system arrive at random. Furthermore, different types of input vary in the amount of work which they demand of the system. As a result, over periods of time on the order of tens of milliseconds, the system may find very little or very much to do. A look at the sequence of Class E visits, as furnished by the simulation indicates that the length of successive Class E revisit times experienced by the system as it processes this random demand on its services

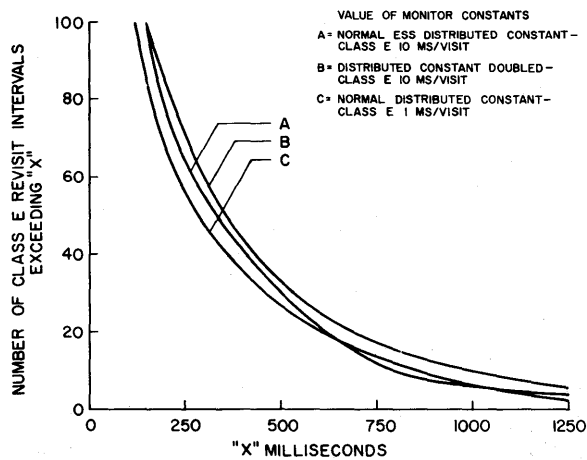


Figure 7. Effect of changes in monitor constants on number of long Class E revisit intervals.

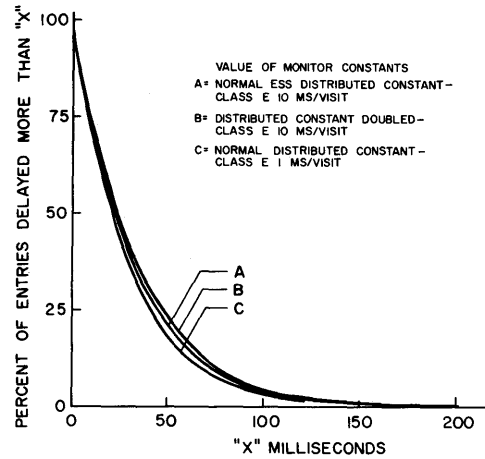


Figure 8. Effect of changes in monitor constants on distribution of Class A hopper entry delays.

are not independent. Unusually long visits tend to be followed by other unusually long visits. Figure 9 illustrates this phenomenon.

Effects of Variable Demand. b) Explanation of Correlation. When an unusually large amount of work arrives, the system stores the work in its hoppers and queues and takes more time to get around to the different classes. When the delay to serving a given class is long, more work can be expected to arrive in the hopper during this long delay, since the work

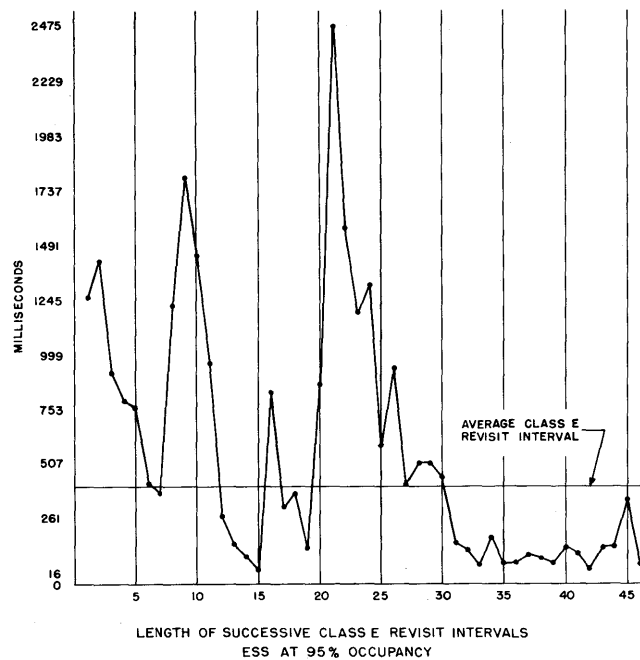


Figure 9. Representative sequence of Class E revisit intervals.

which arrives during this longer period of time will now be stored in the hopper. Consequently when the system finally serves a given hopper it is likely to find more than the usual amount of work and spend more than the usual amount of time in processing this work. When this occurs, the delay in processing the next hopper will therefore be longer than usual. While the system processes this next hopper, work continues to arrive in the remaining hoppers. This leads to a general slowing up of the system in going through its work cycle. Once the system experiences an unexpectedly long interval in any class, it will take some time for it to regain its normal cycling rate. When the system is running at high occupancy, first, the probability of experiencing an unexpectedly long class interval increases, and second, the time taken to recover from such an event is longer.

A brief shortage of work in a class can similarly lead to a series of very short Class E revisit times.

Figure 10 illustrates the system behavior in such a long visit. The scale of time is in milliseconds. The length of time which the system spends serving each class of hoppers is plotted against the horizontal axis. In the upper plot the system revisited Class E very quickly; the next revisit was very long. At the start of the long interval an unusually large amount of work came in. As the system continued to process this work, more than the usual amount continued to enter the system, resulting in a very long cycle. This one long cycle is likely to be followed by another. We concluded that this "snowballing" effect can be controlled if the number of unusually long visits to different classes can somehow be reduced.

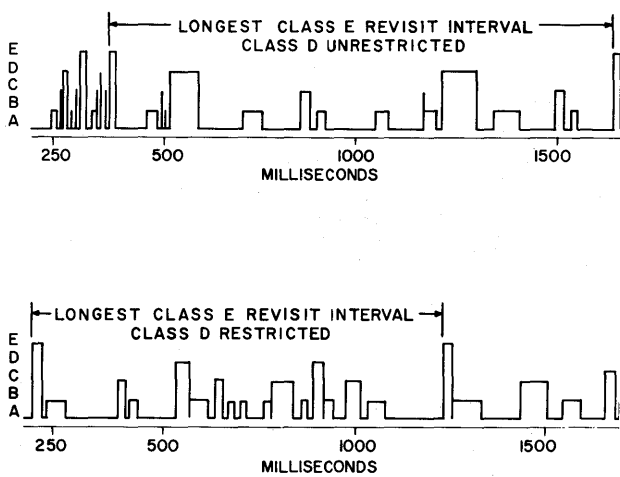


Figure 10. Sequence of time spent in processing work of each class in the monitor work cycle.

In order to verify this inference, it was decided to simulate a monitor which limits the length of time spent in a particular class in the work cycle on any given visit. Although inputs to this class enter at random, only a fixed number will be served on any given visit. This limit does not affect the average rate of serving that hopper; that is, the amount of the traffic previously offered to the system and processed by it is unchanged. Inputs to this one class may encounter a slightly higher delay, but it was expected that they will be served in a reasonable time and not put off indefinitely. Since these inputs constitute a significant portion of the demand on the system, the system could not be operating under the same demand unless these inputs were being accepted at the same average rate as before. Results from the simulation showed this to be the case.

A look at the upper plot in Fig. 10 indicates that long visits occur in a number of classes, among them Class D. It was decided to limit the amount of time spent in some class on any given visit by limiting the number of entries taken from the hoppers of that class on any single visit. Since Class D by definition contains work which can tolerate the most delay, Class D was chosen in preference to Class A.

To study the effect of this limitation, the simulation was run twice with the same input traffic, once without, and then with the limitation in effect. In Fig. 10, the upper and lower plots represent the longest Class E revisit in these respective runs. Allowing the monitor the ability to limit the call processing work served in a particular visit does not change the average E revisit time over a number of minutes. It does, however, eliminate some of the very long and very short visit times and thus reduce the variance on the distribution of the Class E revisit times. Since most of the delays which are likely to exceed the allowable tolerances occur during very long visits, eliminating these very long visits eliminates these intolerable delays.

Figure 11 shows the effect of this limitation on the entire set of Class E revisit times for the two simulation runs previously mentioned. Curves A and A' respectively show the distribution of Class E revisits plotted against the length of visit time for the unrestricted and restricted monitor versions respectively. Curves B and B' show the corresponding percentages of system time which is spent in visits exceeding a certain length. Since the arrival of work is distributed in time, rather than per visit, the second pair of curves permits a better comparison of the

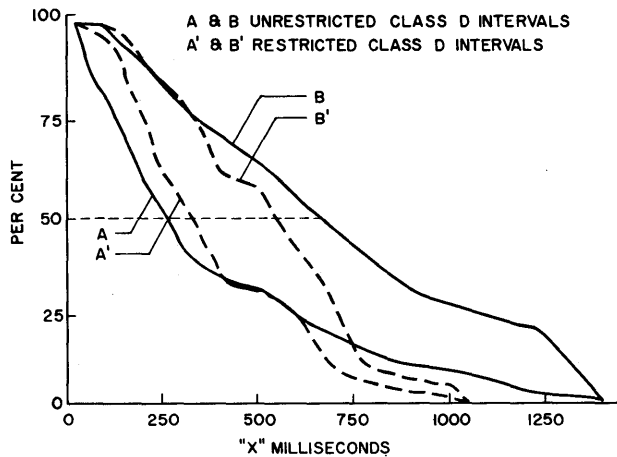


Figure 11. Effects of restricted Class D intervals on system performance.

relative performance of the two versions of the monitor.

In the unrestricted case the system spent half its time experiencing E-E revisit times in excess of 730 milliseconds, while in the restricted case half the time was spent in visit times in excess of 500 milliseconds. Small modifications such as these can materially improve the system performance of the monitor.

An abnormally long time spent on any job or class of jobs will lengthen the delays to other jobs much more than the time used by the job itself. An unusually long period spent on one or a few tasks is likely to result in a series of long processing intervals especially if the average occupancy of the system is high. If a given delay is to be held below some maximum, either the system monitor must exercise some control over the demand on the system, or the system must be operated at a comparatively low occupancy. This implied that both the segmentation of the processing programs and the variations in the input are controlled. If these measures are taken, use of a monitor of the type discussed will permit the system to operate at a comparatively high occupancy and yet meet all delay tolerances.

CONCLUSION

The purpose of the study discussed here was to evaluate and possibly improve a real-time control

monitor. In order to make improvements it was necessary to gain an understanding of the mechanism governing system behavior. Since the system was extremely complex, a detailed simulation of the system was made. This approach has permitted an evaluation of a particular monitor algorithm and has shown how it can be improved. The study has shown this monitor algorithm to be well suited to certain real-time control systems.

ACKNOWLEDGMENTS

In the course of this study most helpful suggestions were made by W. S. Hayward, Jr., J. B. Kruskal, and E. Wolman. G. R. Faulhaber determined the model of the No. 1 ESS which was simulated and suggested the expression for the average Class E revisit time. Miss L. Sadaka planned and wrote most of the simulation program. Miss F. L. Dermond obtained from the ESS program the counts of machine instructions needed to perform various actions. A number of ideas basic to the ESS monitor itself, as well as the implementation of the monitor program in the ESS were contributed by R. B. Smith and S. Silber.

REFERENCES

- The September 1964 issue of *The Bell System Technical Journal* is devoted to various aspects of ESS. It includes the following papers, which are referred to in the text:
1. W. Keister, R. W. Ketchledge and H. E. Vaughan, "No. 1 ESS: System Organization and Objectives," *Bell System Technical Journal*, Sept. 1964, pt. 1.
 2. D. H. Carbaugh et al, "No. 1 ESS Call Processing," *Bell System Technical Journal*, Sept. 1964, pt. 2.
 3. R. W. Downing, J. S. Nowak and L. S. Tuomenoksa, "No. 1 ESS Maintenance Plan," *Bell System Technical Journal*, Sept. 1964, pt. 1.
 4. J. A. Harr, F. F. Taylor with W. Ulrich, "Organization of No. 1 ESS Central Processor," *ibid.*
 5. —, Mrs. E. S. Hoover and R. B. Smith, "Organization of the No. 1 ESS Stored Program," *ibid.*

ON-LINE DEBUGGING TECHNIQUES: A SURVEY

Thomas G. Evans

Air Force Cambridge Research Laboratories, Bedford, Massachusetts

and

D. Lucille Darley

Bolt, Beranek, and Newman, Inc., Cambridge, Massachusetts

INTRODUCTION

One consequence of recent interest in the development of large-scale time-sharing systems to provide on-line computer access to a large number of users has been the widespread realization that the usefulness of such a system is critically dependent on the quality of the software provided to facilitate the interaction between user and machine. In particular, one area of critical importance for effective utilization of such a system is that of facilities for program debugging. In view of the important role they play, surprisingly little attention has been paid to the development of facilities to aid in the process of on-line program debugging. Furthermore, much of the work in this field has been described only in unpublished reports or passed on through the oral tradition, rather than in the published literature. The purpose of this paper is to survey the existing work in this area and discuss some possible extensions to it, with the dual goal of acquainting a wider public with currently-existing techniques and of stimulating further developments.

What, precisely, is the intended scope of this paper? First, we are concerned here only with debugging activities taking place in an on-line environ-

ment, with the user communicating "conversationally" with his computer by means of, typically, a keyboard (or perhaps a display device and light-pen). Inevitably, there exists overlap between on-line and batch-processing debugging techniques, but our concern here is with the former. Second, we are concerned with *program* debugging; one can, of course, view a wide range of computer use as the debugging of something or other; for example, a numerical method or a physical or economic model. On occasion, of course, this line can be difficult to draw, but we intend to restrict ourselves to activities concerned with the discovery and elimination of program "bugs," in the usual sense, from programs written in typical assembly and higher-level languages and to the "subject-matter-independent" facilities provided to an on-line user to assist in this process.

Why do we place such stress on on-line debugging? Is there really so much difference from debugging in a batch-processing mode? Yes, we think so. One can, of course, ignore the conversational aspect of a time-sharing system and treat it as simply a remote-console job-initiation system. However, in doing this, one is neglecting a potentially very powerful tool—the capability (mediated through suitable debugging aids) for a very selective and close control over the exe-

cution of portions of one's program and for the examination of intermediate results, together with the possibility of making on-the-spot changes based on them, as desired. These virtues of on-line access have been praised many times, of course (and debugging is only one activity aided by such access—some on-line uses are so dependent on this type of interaction that they simply have no batch-processing counterparts). We merely wish to add that these benefits for debugging are not automatic results of providing on-line access; as in other aspects of the appearance of on-line systems to their users, careful design of the facilities provided and the conventions for their use pays immense dividends in usability.

Console debugging was common before batch-processing monitors were ever heard of. What's so new about on-line debugging? Nothing, really; current on-line debugging techniques are the result of a gradual development from the days when debugging at the computer console was the norm, as it has remained for small computers over the years. Debugging methods based on single-stepping through parts of a program and on examination and modification of memory registers by means of console lights and switches were the natural precursors of today's more sophisticated techniques, and there is no sharp dividing line at any stage of the progression. Perhaps the critical step was the replacement of console lights and switches by some typewriter-like device as the principal means of communication between user and machine. This permitted the convenient interposition of suitable system programs to facilitate communication between the user and his program. At first they permitted him to examine and modify register contents in typed octal instead of the binary of lights and switches. At a later stage in the development they allowed him to associate symbols with locations in his program and to debug in terms of them, and still later to debug entirely in terms of the original symbols of his assembly-language or higher-level-language programs. The capabilities in this area of current debugging programs will be discussed below. Similarly, a development toward increasing sophistication in the user's control of the flow of his program, as well as in other areas, has taken place and will also be discussed later.

What is the relationship of on-line debugging to time-sharing? On-line debugging (and on-line use of computers in general) is related to time-sharing only in the sense that provision for on-line access to a machine powerful enough for certain classes of

problems may be economically feasible only in a time-sharing environment. Furthermore, it is reasonable to expect that many advances in on-line debugging will arise from the communities of users that have already begun to assemble about the currently-existing large-scale time-sharing systems, as well as from the expenditure on system programming that the existence of such communities makes economically justifiable. However, many of the debugging features we shall be discussing had their origins in work with small machines before the advent of time-sharing systems.

In a survey of on-line debugging, a problem of emphasis arises: one might try to convey some of the flavor of the use of typical currently-available techniques to the reader unfamiliar with any existing on-line debugging system; alternatively, one might try to examine and compare in some detail the most important features of the existing systems. We have resolved the problem by attempting both.

The second section of this paper is devoted to a consideration of the principal features of past and present on-line debugging systems known to us, together with some remarks on implementation, on use of displays, and on some implications of the requirements of debugging systems for compiler construction and for hardware. We make no claim of exhaustive coverage. We have discussed those systems which incorporated features which seem to us to have been interesting or significant contributions to the present state of development of on-line debugging.

The third section attempts to impart some "feel" for current on-line debugging methods through two annotated examples. One represents a session devoted to debugging a program written in a typical (but nonexistent) assembly language; the other, a program written in a representative (also nonexistent) algebraic-type language. The examples are idealized in that no one present system contains all of the capabilities illustrated (or uses precisely the set of communication conventions we have adopted), but every feature shown is present in some existing system.

The concluding section contains a few final comments of a general nature.

SURVEY OF EXISTING SYSTEMS

Assembly-Language Debugging

We shall first consider facilities to aid in the debugging of programs written in assembly language.

We have made no extensive effort to disentangle all the threads of the earliest efforts at developing typewriter-based debugging programs. However, the early program which had the greatest influence on subsequent developments was that of Gilmore¹ for the TX-O computer at Lincoln Laboratory in 1957. It was the first in a series of closely-related and successively more elaborate debugging programs, including UT-3² and FLIT³ for the TX-O (after it was moved to MIT), and DDT^{4,5} for the PDP-1 at MIT. FLIT, in particular, was a notable accomplishment, embodying capabilities on which much subsequent work with on-line assembly-language debugging has been based. With FLIT, for the first time, it was possible for the user to examine and modify his program in terms of the symbols used in his source program and, in fact, to examine and change the contents of registers in a form almost identical to that used in the corresponding assembly language. Furthermore, while some earlier typewriter programs had permitted one-instruction-at-a-time tracing of a program, by analogy to the console single-step switch familiar to their creators, FLIT introduced what is perhaps the central notion of interactive debugging, that of a user-controlled breakpoint. This technique, which we shall see illustrated in both assembly-language and algebraic-language debugging in a later section ("Examples—Two Debugging Sessions"), consists of permitting the user to specify (symbolically, typically) a point or points in his program at which he wishes to interrupt its flow and return to the debugging routine, which at entry stores the state of the live registers to permit subsequent continuation from the breakpoint, then permits the user to examine the state of his program at that point and make changes, if he wishes, before continuing. All that is required is that the debugging program save the user's instruction at the desired breakpoint location and plant in its place a suitable transfer to itself. The effectiveness of the technique is dependent, of course, on the ease to the user of placing and removing breakpoints and on the quality of the facilities for examination and modification available to him while at a breakpoint. With judicious use, the breakpoint can be a very flexible tool, giving the user great selectivity in the degree of fineness of his examination of a portion of a program. In the hands of an experienced user, it can permit quite rapid isolation of many types of program error. Here, as in other aspects of on-line work, convenience is critical. The user with only "examine and

modify" capabilities available to him could, of course, get the effect of breakpoints by inserting transfer instructions to appropriate inserted code, but the convenience and freedom from elaborate bookkeeping so important to the "iterative" use of breakpoints described above are lost.

FLIT was a program for a one-of-a-kind machine, the TX-O. Consequently, it never became well-known outside its user community at MIT. It was through DDT (written at MIT soon after FLIT as its counterpart on the PDP-1 and embodying much the same set of capabilities, including those sketched above) that these notions were extensively spread about as the PDP-1 became a relatively widely used machine. In this way, FLIT and DDT became the acknowledged source of a large portion of the assembly language debugging programs in the major currently operating time-sharing systems possessing such facilities.

One of the most important characteristics of FLIT and DDT was the care devoted to the design of the typing conventions. Single-letter commands and a structure in which frequently desired states could be reached easily from the present one (e.g., look at the contents of the current register ± 1 , look at the contents of the register addressed in the current register) minimized typing and aided rapid interaction. Similarly, convenient ways of typing the contents of a given register in alternate formats (e.g., symbolic, decimal, octal) were provided.

Starting with these capabilities, extensions have been made in a number of directions in more recent work. We shall discuss some of these. With the capability for input of machine instructions in symbolic assembly-language form, DDT is already nearly an "on-line assembler," suitable as the sole tool for on-line writing and testing of small programs. With this use in mind, Edwards and Minsky⁶ added an "undefined symbol" capability to DDT. In conventional DDT, input of a line of code involving a symbol not already defined by the user results in an error message. In their version, it results in a special symbol table entry. Such entries are linked together, and when the symbol is ultimately defined by the user its previous occurrences are filled in appropriately. This capability has also been included in the assembly-language system⁷ of the Berkeley time-sharing system (SDS 940).

DDT permits the user unlimited freedom to patch his program arbitrarily by inserting whatever he likes in some available space, then planting a transfer to

this insertion in his program wherever he desires. This very freedom, unfortunately, can lead to situations in which debugging of complex programs ultimately bogs down in a morass of patches on patches. Furthermore, even when a highly patched program has finally been made to perform satisfactorily, the road to a corresponding "cleaned-up" symbolic version of the program can still be a very long and error-susceptible one. We know of two efforts to incorporate at least partial solutions to these book-keeping problems into assembly-language debugging systems. In one approach, followed by Lampson⁸ in the design of one version of the assembly-language debugging facilities in the Berkeley system, the user requests the insertion of a specified piece of symbolic code starting at a specified symbolic location in his program (or deletion of a portion of the existing program, or both). In response to this request, the debugging program performs two distinct activities:

1. It edits the user's changes into his symbolic program stored on the drum.
2. It assembles the user's addition into a "patch area" of core and automatically links the resulting code to the user's program in a straightforward way by copying instructions and inserting transfers, as necessary.

Thus, at each stage of the debugging process, the user's patched binary program in core is "computationally equivalent" to the edited version of his symbolic on drum. At the completion of the debugging session, the user's updated symbolic is stored again among his files.

An earlier approach⁹ to the same problem, taken by the present authors in work with an assembly-language debugging system for the M-460 computer at Air Force Cambridge Research Laboratories, is quite different in implementation. Once again, the on-line user presents insertions, deletions, or a mixture of both (again in symbolic assembly language) to the debugging program, using a quite similar set of conventions. Once again two actions are taken:

1. The symbolic changes are stored in a form suitable for entry, along with the original symbolic program, to an editing program at the end of the debugging session. This is automatically done and the user provided with an updated symbolic file. This difference—saving the corrections to do all the editing at one time vs editing for each correction,

as in the system discussed previously—is a thoroughly trivial and inessential one.

2. Instead of a patch being made corresponding to the user's change, the part of the program affected by the change is relocated appropriately in core. If the change is an insertion, for example, the new code is assembled into the space left vacant by the relocation of the program from that point on. This relocation process is possible only because the relocation information resulting from the assembly of the user's program, in addition to being used by the relocating loader, is collected by it into a list structure which is used by the debugging program each time a program change is called for by the user, then updated accordingly. The symbol table passed by the assembler to the debugging program must also be updated each time. Thus the idea of "patching" disappears completely. This relocation process can be rather time-consuming on large programs, but has certain compensating advantages over the (quite fast) "automatic patching" approach of Ref. 8. In particular, it avoids the two drawbacks of his system listed by Lampson:

- a. In situations in which location of words in core relative to each other is important (for example, subroutine calls picking up arguments from following locations), the patched binary and the edited symbolic may behave differently.
- b. The automatic patching process leaves core in a rather confusing state, which may require relatively frequent reassembly for readability. For example, the user who wishes to insert a breakpoint at an instruction inserted during one of his previous modifications must track down the present location of that instruction by finding and following out the patching. Thus, much of the advantage of automatic patch insertion could well be nullified.

Any evaluation of the two approaches must balance the added program complexity and computation time required by the relocation approach against the possible cost in inconvenience to the user of the above difficulties (or, alternatively, the cost in computation time of the additional assemblies that may be required to preserve sufficient readability).

One further extension to DDT in more recent work pertains to the use of breakpoints. In addition to the flexibility in the placement and moving of

breakpoints which is already present in DDT, a facility has been added in a number of debugging programs (including those for the SDC time-sharing system,¹⁰ the DEC PDP-6,¹¹ and the M-460 at AFCRL) permitting the user to make the breakpoints conditional; when the breakpoint location is reached, some test previously supplied on-line by the user is executed to determine whether the break is to be made (that is, control turned over to the user) or whether execution of the user's program should continue. This technique permits still greater selectivity; the user can run his program till some specified condition prevails at a specified point, then examine the program state in whatever detail he wishes. The SDC system gives the user a choice of a number of built-in conditions; the other two permit the user to insert an arbitrary piece of assembly-language code as the break test associated with each breakpoint. Ideally one would like to combine "canned," easily specifiable tests for certain common situations with the capability of writing arbitrary tests when desired. DDT, incidentally, had a rudimentary but often useful form of the conditional breakpoint which has been preserved in several later systems; upon insertion of a breakpoint, the user may specify (simply by preceding the command with a number n) that the break is not to occur until the n th time that that point is reached in the execution of the program.

A possibility we have not yet examined, but which forms a basic tool of some early on-line debugging programs, is that of instruction-by-instruction tracing. More sophisticated versions of such tracing, with considerable flexibility available to the user, have been incorporated in debugging packages for batch-processing use, but such tracing features have typically been omitted from more recent on-line systems in favor of the breakpoint, on the grounds that tracing represents a failure to make the most of the capability for intensive interaction possible in such a system and, at best, tends to produce considerable irrelevant printout, a serious consideration for an on-line user. However, it seems to us reasonable to provide some tracing capabilities in an on-line system, especially since they can share much of the machinery already provided for breakpoints. The user should be able to specify a location in his program and ask either for the printing of certain information, for control, or for a combination of both whenever that program point is reached (and a specified condition is satis-

fied). Currently no assembly-language debugging system appears to have quite this full capability, though PDP-6 DDT¹¹ and the SDC DEBUG package¹⁰ are close. Both are limited in the amount of information that can be specified in advance to be printed at a break—in the PDP-6 DDT to one register and in DEBUG to one register or a live register dump or a dump of some block of registers. Furthermore, as mentioned above, DEBUG does not permit the composition of elaborate conditions for a break to occur.

Another desirable feature not widely found in current assembly-language debugging systems is extensibility, in the sense of the capability for conveniently defining complex debugging operations in terms of the available primitives. The most general existing facility of this type appears to be that described in Ref. 8, where the macro-expansion capability of the assembler used to process input to the DDT lends itself quite naturally to this purpose.

Programs of the DDT family have many useful features in addition to the ones we have described. As one example, it is typically possible to conduct a search between specified limits in core for all words matching a given word in the bits specified by a given mask.

Higher-Level-Language Debugging

When we turn to the examination of on-line debugging facilities for programs written in higher-level languages comparable to those we have considered for assembly-language programs, we find that, broadly speaking, a close analog of almost every principal assembly-language debugging technique exists in at least one debugging system pertaining to some higher-level language. However, on-line debugging facilities for higher-level languages are in general less well-developed and less widely used (relative to the use of the languages) than their assembly-language counterparts. In part, this situation is probably a consequence of the wide diversity of languages in this class; probably it is still more a result of the fact that the small machines on which the assembly-language techniques originated and were cultivated were typically considered too small to support higher-level language compiling systems and were programmed almost exclusively in assembly language. Thus work in on-line debugging of higher-level languages is of comparatively recent origin. We shall be examining debugging systems for relatively few

languages in relatively few on-line computing systems. This is not to say that much more on-line debugging (in the sense that the user at a remote console starts his program, examines the final results or diagnostics in essentially the manner of batch processing, edits his program, and tries again) is not taking place in these and other systems with these and other higher-level languages. However, we are concerned especially with efforts to obtain systems which permit the on-line user something like the flexible control over the execution of his program and the capability of examining and modifying it that are available to the one-line user of the assembly-language debugging aids we have discussed.

The language for which perhaps the most effort has been expended in the development of on-line debugging aids is the list-processing language LISP 1.5. However, no discussion of these debugging features has appeared in the literature; they are far from completely described even in internal memoranda. The first two full-scale on-line implementations of LISP were those for the MAC system^{12,13} (a modification of the batch-processing LISP system for the IBM 7094 to run under the MAC time-sharing system) and for the M-460 computer at AFCRL. Subsequently, on-line LISP 1.5 systems have been created for the SDC time-sharing system,¹⁴ the Berkeley system,¹⁵ the DEC PDP-6,¹⁶ and the DEC PDP-1 at Bolt, Beranek, and Newman, Inc.¹⁷ We shall discuss only the debugging features of the MAC and M-460 systems, as the later systems contain essentially no debugging aids not already present in these.

First, the extensive tracing facilities of the LISP system were made accessible to the on-line user. Later, they were extended and made conditional in both systems. An editing program—not a conventional text editor but a program permitting the user to modify the list structure in which LISP functions are stored for interpretation—was introduced by Martin into the MAC LISP system and soon modified for use in M-460 LISP. This editor proved to be a powerful tool, permitting quite easy program modification in many cases. Conditional breakpoints (insertible at any point in a LISP function definition) were introduced into the M-460 LISP system by one of the present authors—apparently, along with the introduction of breakpoints into the SDC IPL-V system by Weissman,¹⁸ their first use in higher-level language debugging—and soon after incorporated in MAC LISP. Conditional breakpointing and tracing have proved quite powerful for LISP

debugging, as it is possible to use the full capability of the LISP language for the on-line composition of the conditions. Thus one can easily express an elaborate logical condition for which the counterpart in assembly language might be quite complex. Furthermore, by “canning” a few useful special predicates for use in writing conditions, even more selectivity in suppressing irrelevant tracing and breakpoints can be attained. For example, in M-460 LISP there is a machine-language LISP function which examines the interpreter's pushdown list to answer the question: “At this point in the execution of the program are we inside a call to function X?” Incidentally, the ability of LISP to handle recursion has proved very useful in debugging—the full capability of the LISP system is available at a breakpoint inside a function being executed. With some care, it has been possible, for example, to find a bug while at a breakpoint in running a test case, call the editor to make a correction, run the program on a simpler test case to verify the correctness of the change, then resume execution of the original test case from the breakpoint (without the addition of any special machinery to the system for saving and restoring a program state, etc.).

At this point, it should be mentioned that both LISP systems mentioned contain both an interpreter (of LISP functions stored as list structures) and a compiler (of LISP functions into machine code), and that interpreted and compiled functions may be quite freely intermixed. The existence of the interpreter made the implementation of the debugging facilities described above relatively simple. For example, insertion of breakpoints at arbitrary locations in a LISP program is readily implemented by modifying the list structure corresponding to the program so as to call the breakpoint-handling routine appropriately. In addition, interpretation has the advantage that various types of user errors may be conveniently detected at run time. A further advantage in this case is that LISP is precisely a language for manipulation of list structures so the breakpoint insertion routine, among others, could itself be written entirely in LISP. On the other hand, the feature in LISP that permits tracing of entries (with printing of arguments) and exits (with printing of values) of specified routines applies to both compiled and interpreted routines. However, the usual mode of operation in the systems mentioned has been to debug interpretively, then compile the debugged programs in cases where the great speed advantage to

be gained by compiling is important. In general, interpretation presents similar advantages for other higher-level languages, and we shall see below that it has furnished the basis for on-line debugging systems for other languages, as well. We shall also mention two systems which work with a compiled program. Then we shall consider one effort to design a system combining interpretation and compilation, with the intention of combining the speed advantage of compiled programs with the ease of modification that comes with interpretation.

The well-known QUIKTRAN system¹⁹ is based on interpretation of FORTRAN statements. The FORTRAN program under debugging may be modified freely by insertion and deletion of statements. A form of unconditional breakpoint capability is included in the sense that a statement can be inserted at any point in the program which, when reached, has the effect of transferring control to the user. Capability for examining and modifying variables is present, as well as a variety of modes of tracing (print all assignments to variables in a given portion of the program, all assignments to selected variables, all control transfers within a specified region, etc.). Furthermore, extensive run-time diagnostics made possible by the interpretive mode are provided, and several unusual "bookkeeping" features, similarly based on interpretation, are available, such as the AUDIT command, which generates information as to which portions of the program were never executed, which variables were never set, or set but never used, during a given execution of the program.

Another on-line debugging system based on interpretation is that for IPL-V in the SDC time-sharing system.¹⁸ It contains (nonconditional) breakpoint and tracing capabilities similar to those sketched above for LISP.

The FORTRAN debugging system²⁰ for the Berkeley time-sharing system and the MADBUG system²¹ for the debugging of MAD language programs are very similar in their debugging capabilities, though different in overall scope: MADBUG contains a set of editing facilities as well, while editing of FORTRAN symbolic programs is carried out in the Berkeley system by use of a general-purpose editing routine present in the time-sharing system. In both cases debugging is performed on a compiled version of the program, and the user can readily ask for the values of variables and change them. Breakpoints (nonconditional, as it happens) at any specified statement (in the Berkeley FOR-

TRAN, labeled statement) may be inserted and deleted. However, no facility is provided to modify portions of the user's program (in both systems, a user familiar with the code produced by the compiler could, of course, use the available assembly-language debugging facilities to make such local modifications). The only way to make program changes is to edit the symbolic version and recompile the whole program.

The notion of an "incremental" compiler, in which only those portions of a program to be changed need to be recompiled, has been frequently discussed; Lock²² at California Institute of Technology has given a detailed sketch of the design of a system with such capabilities. The notion is to compile each program statement separately and place the resulting code, together with a copy of the symbolic form of the statement and certain pointers and other information, depending on the type of statement, in a contiguous block of core. These blocks would be linked together in lists. Since the language in question is ALGOL, in which "statement" is a recursively defined concept, one has a list structure (lists with elements which are lists, etc.) instead of the one-level list of statements one would have with, for example, FORTRAN. Insertion and deletion at the statement level proceed straightforwardly by modification of this list structure. Control is returned to a monitor between statements (this is a property of the code generated for each statement) permitting, among other things, breakpoint capability at the statement level (though the author proposes simply a single-statement mode of operation modeled, apparently, on single-step-switch machine-language debugging). The scheme is interesting and quite ambitious. It remains to be seen whether the organization based on compiled statements with interpreted flow of control between them leads to significantly faster execution times than pure interpretation with a well-designed internal representation such as that of QUIKTRAN. One possible modification in the scheme would be to arrange things so that the code in the block corresponding to a statement transfers, not back to the executive, but to what at that point is the correct next statement. The executive would maintain a table of these transfer locations and "breakpoint" them, so to speak, whenever this was called for (as a result, for example, of a breakpoint request by the user). Thus, at the cost of some additional complexity in the executive, almost all the speed advantage of full

compilation would be realized with no loss in the capabilities available to the user.

It seems appropriate to mention at this point a class of languages of which JOSS, BASIC, and TINT are the best-known examples, even though a principal characteristic of these languages, especially the first two, is their lack of anything which looks like the tracing or breakpoint features we have discussed. These are "small" languages designed precisely for easy learning and convenient on-line use for problems requiring a numerical computing capacity somewhere between a desk calculator and the typical "FORTRAN + large computer" installation. In all three languages, insertion, deletion, and modification of statements is extremely easy; since this is so, the effect of tracing and breakpoints can be achieved for the small, relatively simple programs in question by insertion of print and halt statements respectively. Thus, at the borderline of the class of languages and associated debugging tools that we have discussed earlier we find a class of languages that have rather effectively transcended the need for such tools by careful design and ruthless simplification of language structure corresponding to the setting of limited objectives for the range of usefulness of the language.

Hardware Aspects

There are many points of interaction between computer hardware design and the design of software debugging facilities. We shall mention just two:

1. The capabilities of user consoles have a great impact on the range of debugging facilities. Suppose the user is provided with a display device in addition to (or instead of) his keyboard. One may use this added capability relatively conservatively as an extension of facilities already present. For example, the Edwards-Minsky version of DDT already mentioned⁶ used a display to permit the user much more rapid and convenient examination (in symbolic and octal) of his program in core than would have been feasible with a typewriter alone. Programs to display core in octal already existed more than 10 years ago.²³ Other, more radical uses of display devices in debugging are now being investigated. Flow-chart languages, where programs are created on-line by generating a flow chart with a light-pen, are being studied at Lincoln Laboratory²⁴ and at RAND.²⁵ A dynamic display of the program state at any point in terms of the flow chart is expected to be a useful

debugging tool. Other work at Lincoln Laboratory²³ is directed toward dynamically mapping out on a display device the behavior of a more conventionally-constructed program by means of a flow diagram, which is again expected to be a useful debugging aid.

2. Another area of contact between hardware and debugging is involved with trapping. Program-controllable facilities for trapping on certain machine conditions give promise of being a very important debugging aid. The TX-2 computer at Lincoln Laboratory, for example, has recently been provided with a quite powerful interrupt system of this nature, which has been made accessible to the on-line user through commands to a DDT-like program.²⁶ The user may ask for a trap on any combination of a number of conditions, such as a store into a specified register, execution of an instruction at a specified location, or execution of any skip or jump instruction. The debugging program handles the interrupt and reports the relevant information to the user.

EXAMPLES: TWO DEBUGGING SESSIONS

Assembly Language Debugging

Assume we wish to debug the following program (written in a typical—but mythical—assembly language), which is meant to perform a simple-minded exchange sort on a table of five numbers.

```

sort      pze
          call    .readf
          bci     data1
          pze     table
          lix     nm2,1
loop      lda     table+1,1
          sub     table,1
          sma
          jmp     ok
          lda     table,1
          ldq     table+1,1
          sta     table+1,1
          stq     table,1
ok        tiz     .+2,1
          jmp     loop
          ret     sort
nm2       pze
table     bss    5
          end

```

(This is admittedly a trivial program which should not need the elaborate debugging facilities we have discussed; however, it should serve to illustrate the application of these techniques in an otherwise reasonably realistic context.)

We assume that, previous to this debugging session, we have stored our symbolic program as a file, either by reading in cards or paper tape, or by typing the program in directly from our console. We then assembled our program from the file and created a new file containing the loadable form of the program as well as the symbol table. We omit describing these procedures in detail, for the process of controlling an assembly on-line and the error diagnostics received are much the same as assembling off-line (with, however, the advantage that any errors detected by the assembler can be corrected immediately). We also assume that a test case file called data1 (which will be read by our program) has been written. For definiteness, assume it consists of the numbers 3, 5, 2, 1, 4 in that order.

The loading system has brought our program into core and has left us in contact with the debugging system, which it has supplied with the symbol table for our program. We immediately attempt to execute the program by typing:

G sort

(This calls sort, which is written as a subroutine.) The debug system responds with a carriage return, indicating completion of our program. We type:

P table; table+4

which prints:

```
table  3
        5
        2
        1
        4
```

That is, the table we input is unchanged. Examining our program, we note that the instruction at ok performs a test for the end of a pass through the table. It seems a plausible instruction to monitor, so we insert a breakpoint (number 1) there:

B1 ok

and then execute the program again. The computer responds with:

ok

indicating that it has reached the breakpoint. At this point we can examine whatever registers we wish,

including live registers, by typing the symbolic name plus a tab. The computer will respond with the contents of the register in symbolic format, tab, and wait for us to modify the contents of the register. If we don't wish to, a carriage return signifies this. Index register 1 is important in our program, so we examine it:

```
I1      0
```

We see that this value is incorrect. The instruction at loop-1 supposedly loads index register 1. We check it:

```
loop-1  lix nm2,1
```

This is apparently correct, so we check nm2:

```
nm2      0
```

This is our error. We neglected to initialize nm2. We give the following command:

```
C nm2
nm2      oct 3
```

which says to change the contents of the register labeled nm2 to whatever follows; in this case, the register is to carry the same label but contain the number 3, the length of our table minus 2. Our program is physically changed in core, and the necessary information concerning this change is saved so it can be given to an editing program at the conclusion of our debugging session. We remove the breakpoint inserted earlier by typing:

B1

and then start the program again. Again the debug system carriage returns. We now check the contents of the table, as before:

```
table  1
        3
        5
        2
        4
```

Obviously not all of the ordering is correct. Perhaps it would be useful to reinsert the breakpoint at ok, since it is the instruction immediately following the instructions that switch the contents of registers. However, we would like to break here only if an exchange did occur, and at the break we would like to print the contents of the two registers in the table that were switched. This allows us to monitor the successive changes in the table, so we can see at

what point something goes wrong. We insert this type of breakpoint by:

```
B1 ok: P table,1; table+1,1: C
B1C
lda    table+1,1
sub    table,1
spa
end
```

The first line indicates that breakpoint 1 should be inserted at ok and when that point is reached two things should be printed out: the contents of the register at (table + the contents of index register 1), and the register at (table+1 + the contents of index register 1). C means to continue after the printing without transferring control to the user. The second line indicates that we are going to give a condition for breakpoint 1, and the next three lines are the condition, with the instruction following the spa (skip on positive AC) the break branch and the instruction following that the proceed branch. With each breakpoint the debug program associates three registers that are used in determining if a break should occur when the breakpoint has been reached. Initially, and each time a breakpoint is removed, the three registers appear as:

```
nop (no operation)
(return for break to occur)
(return for no break to occur)
```

With this arrangement, a break occurs each time the breakpoint is reached. When a condition (other than a single skip instruction) for a break is entered, as we did above, the nop instruction is automatically changed to a jump to a patch region where the code we supply is inserted. The first register following this code is assumed to be the break condition and a jump is inserted there to the first return. The second instruction following the code is set up as a jump to the second return.

We now execute our program again and get the following results:

```
ok
table+2    1
table+3    2

ok
table+1    1
table+2    5
```

```
ok
table      1
table+1    3
```

which is correct as far as it goes, but after the above printout the debug system carriage returns, again indicating that our program has returned. Looking at our program again, we see that we left out the outer loop in our coding and are making only one pass through the table. We make the following changes:

```
I sort+3
loop2     stz switch
I loop+3
          idx switch

I ok+1
          lda switch
          sza
          jmp loop2

I nm2
switch    pze
```

The first change inserts an instruction labeled loop2 after sort+3 to initialize the register labeled switch, which is at this point undefined. The second change inserts an instruction after loop+3 to increment the contents of switch. The third change is to insert three instructions after ok+1. They again refer to switch and also to loop2, which was defined in the first change. The fourth change defines switch, and at this point all references to it are automatically filled in.

We now run our program again after removing the breakpoint and type out the results as above. This time the table is fully sorted. At this point in our debugging session we decide that we now have a working program in core. To get a clean symbolic version, we give a command to the debug system to supply the changes which we have made (and it has kept) to an editing program, along with our original file. The editing process takes place and the new file is written and given whatever name we have specified. Our final (purportedly debugged) program looks like this:

```
sort      pze
          call .readf
          bci data1
          pze table
loop2     stz switch
          lix nm2,1
loop      lda table+1,1
          sub table,1
```

```

sma
  jmp    ok
  idx    switch
  lda    table,1
  ldq    table+1,1
  sta    table+1,1
  stq    table,1
ok      tiz    .+2,1
        jmp    loop
        lda    switch
        sza
        jmp    loop2
        ret    sort
nm2     oct    3
switch  pze
table   bss    5
        end

```

Higher-Level-Language Debugging

Our example of on-line debugging of a higher-level-language program will be shorter than the preceding assembly-language example, since we simply wish to show that the facilities exhibited there for control of program flow and for examination and modification of program and data have their counterparts at other levels of language as well. Correspondingly, our program example is even more trivial; it is the same exchange sort programmed in a typical (but again mythical) algebraic language with the same sort of (admittedly implausible for a program of this simplicity) errors.

Again we assume our program has previously been made into a symbolic file, then compiled and loaded. We shall test it on the same file (data1) as we used in the previous example. Our program reads as follows:

```

program sort;
  array table (5);
  readfile(data1,table);
loop: for i←1 step 1 unit 4 do through last;
  if table(i) ≤ table(i+1) go to last;
  begin table (i) ← table(i+1);
    table(i+1) ← table(i)
  end;
last: continue;
  go to loop;
finish

```

(readfile is a system routine that we call to fill the array named table from the file named data1). We

run our program (named sort) by typing G sort, as before. In this case, we conclude after a respectable interval that our program is looping. By striking the interrupt key, we return to the debugging program. We realize that we have failed to provide a test to escape from the program after a pass through the table generates no exchanges. We therefore make several additions to our program, as follows:

```

I loop - 1
  ;init: switch ← false;

```

(which means insert the labeled statement setting the Boolean variable switch to false after statement loop - 1, that is, then "readfile" statement).

```

C last + 1
  ;if switch then go to init else exit;

```

(which means insert the statement testing the variable switch instead of the statement last + 1, that is, "go to loop"). And finally:

```

I loop + 2, 2
  ;switch ← true;

```

(which means insert the statement setting the variable switch to true after the second statement of the compound statement at location loop + 2).

To verify the last change, for example, we can type

```
E loop + 2
```

which prints:

```

begin table (i) ← table(i+1);
  table(i+1) ← table(i);
  switch ← true
end

```

Now we try running our program again. This time it terminates and returns control to the debugging program. However, when we examine the results by typing:

```
P table(1);table(5)
```

we get:

```

table(1) = 1
table(2) = 1

```

At this point we interrupt the printout, since our answers are clearly in error. We insert a breakpoint at the statement labeled last and add the condition that we break only if switch has been set to true, by typing:

```

B1 last
B1C switch

```

We run the program again and get the breakpoint printout:

last

We examine the indexing variable:

i 2

So we look at:

table(2) 1
table(3) 1

This tells us we're doing the exchange wrong. We see that we are destroying table(i) too soon, and correct this by typing:

```
I loop+2,0
;tem←table(i);
```

and

```
C loop+2,3
;table(i+1)←tem;
```

and rerun our program. This time, when we examine table, it is properly ordered. We terminate the session, as before, by passing the accumulated corrections to the editing program, which updates our symbolic. The final version of our program looks like:

```
program sort;
      array table(5);
      readfile(data1,table);
      init: switch←false;
      loop: for i←-1 step 1 until 4 do through last;
            if table(i) ≤ table(i+1) go to last;
            begin tem←table(i);
                  table(i)←table(i+1);
                  table(i+1)←tem;
                  switch←true
            end;
      last: continue;
            if switch then go to init else exit;
finish
```

Once again, in conclusion, we stress that the programs used in the examples of this section were not intended as representative of those for which such on-line debugging facilities are necessary or even appropriate, but rather as uncluttered vehicles for some simple illustrations of the use of these facilities.

SOME FINAL REMARKS

Very little data seems to exist on the relative efficiency of on-line program debugging versus de-

bugging in a batch-processing mode, though Ref. 27 represents a first effort in this direction and will presumably be followed by others. Meanwhile, we can only record our subjective impressions of a quite widespread enthusiasm for the utility of on-line debugging facilities among those with whom we have discussed the subject.

What are some criteria for a good interactive debugging system (for an experienced user)? We shall try to abstract some (perhaps platitudinous) principles from the wide variety of systems considered above:

1. The user must have flexible control over the execution of his program. He must be able to specify this control in terms of the natural units, small and large, of the language in question and be able to carry this control down to the finest level of detail, if required (a single instruction in assembly language, or single noncompound statement in an ALGOL-type language).
2. The user must be able to examine and "incrementally" modify both data and program at any time and do so in terms of the notation of the language of the program.
3. The conventions of the debugging control language should be designed to minimize typing and should convey information to the user as concisely as is compatible with rapid comprehension.
4. Automatic updating of a user's symbolic file "in parallel" with modification of the in-core representation of his program should be possible, to eliminate a distinct separate phase of cleanup of the symbolic and re-debugging.

Each of these capabilities is now present, as we have seen, to some degree in some current systems. With feasibility thus demonstrated, in the near future we shall presumably see the integration of features taken from these systems into comprehensive on-line debugging systems possessing all the desirable characteristics listed above. Incidentally, this seems to present an opportunity for some valuable voluntary standardization; if the appearance to the user of the debugging system for a given language could be made the same over a number of future time-sharing

systems (at least to the degree that the language in question is itself standardized), considerable savings could well be realized. At any rate, this would seem to be an opportune time to consider the possibility.

In addition to consolidation of known techniques into comprehensive, widely available systems, one can also expect the development of a variety of new approaches; in particular, we have mentioned research which seeks to exploit the full capabilities of displays for debugging, as well as the potential value for debugging of flexible programmable interrupt capabilities in computer hardware.

Considerable interest has been shown in recent years in the development of methods for proving that a given computer program has certain properties. If this avenue of research proves successful, we may one day see the virtual elimination or at least diminution in importance of the program debugging process. Until then, debugging will remain a critical phase and potential bottleneck in the effective utilization of computers. It has been suggested²⁸ that, from a period in which the limitations on computer use were in core size and in sheer lack of enough processor cycles to go around, followed by one of lack of adequate languages, we are now entering an era in which computer use is "debugging-limited." If this is so, the development of improved on-line debugging facilities would seem to be a particularly fruitful and valuable endeavor, as well as a quite fascinating one.

REFERENCES

1. J. T. Gilmore, "TX-O Direct Input Utility System," Memo 6M-5097, Lincoln Laboratory, MIT (Apr. 1957).
2. C. Woodward, "UT-3: A Direct Input Routine for TX-O," Memo M-5001-1, Dept. of Elect. Eng'g., MIT (July 1958).
3. T. G. Stockham and J. B. Dennis, "FLIT—Flexowriter Interrogation Tape: A Symbolic Utility Program for TX-O," Memo 5001-23, Dept. of Elect. Eng'g., MIT (July 1960).
4. R. Saunders and R. Wagner, "On-Line Debugging Systems," *Proc. IFIP Congress, 1956, Vol. 2*, Spartan Books, Washington, D.C.
5. A. Kotok, "DEC Debugging Tape," Memo MIT-1 (rev.), MIT (Dec. 1961).
6. D. J. Edwards and M. L. Minsky, "Recent Improvements in DDT," AI Memo #60, MIT (Nov. 1963).
7. L. P. Deutsch and B. W. Lampson, "DDT Time Sharing Debugging System Reference Manual," Document #30.40.10 (rev.), Univ. of Calif., Berkeley (May 1965).
8. B. W. Lampson, "Interactive Machine Language Programming," *Proc. FJCC*, 1965.
9. T. G. Evans and D. L. Darley, "DEBUG—An Extension to Current Online Debugging Techniques," *Comm. of the ACM*, vol. 8, no. 5 (May 1965).
10. R. R. Linde, "Q-32 Time-Sharing System User's Guide Executive Service: Debugging (DBUG)," TM-2708/390/00, Syst. Devel. Corp. (Apr. 1966).
11. "PDP-6 DDT Manual," Digital Equipment Corp., 1965.
12. W. Martin and T. Hart, "Time-Sharing LISP," Memo MAC-M-153 (rev. 1964).
13. W. Teitelman, "EDIT and BREAK Functions for LISP," Memo MAC-M-264, MIT (1965).
14. S. L. Kameny, "LISP 1.5 Reference Manual for Q-32," TM-2337/101/00, Syst. Devel. Corp. (Aug. 1965).
15. L. P. Deutsch and B. W. Lampson, "Reference Manual—930 LISP," Document #30.50.40 (rev.), Univ. of Calif., Berkeley (Nov. 1965).
16. P. Samson, "PDP-6 LISP," Memo MAC-M-313, MIT (June 1966).
17. D. G. Bobrow et al, "The BBN-LISP System," AFCRL-66-180, Bolt, Beranek, and Newman, Inc., Cambridge, Mass. (Feb. 1966).
18. J. E. Schwartz, E. G. Coffman, and C. Weissman, "A General-Purpose Time-Sharing System," *Proc. SJCC*, 1964.
19. T. M. Dunn and J. H. Morrissey, "Remote Computing—An Experimental System," *ibid.*
20. C. S. Carr, "FORTRAN II Reference Manual," Document #30.50.50, Univ. of Calif., Berkeley (Feb. 1966).
21. R. S. Fabry, "MADBUG—A MAD Debugging System," in *The Compatible Time-Sharing System, A Programmer's Guide*, 2d ed., MIT Press, Cambridge, Mass., 1965.
22. K. Lock, "Structuring Programs for Multi-program Time-Sharing On-Line Applications," *Proc. FJCC*, 1965.
23. T. G. Stockham, "Some Methods of Graphical Debugging," to appear in *Proc. IBM Scientific Computing Symposium on Man-Machine Communication* (held May 1965).

24. W. R. Sutherland, "On-Line Graphical Specification of Procedures," presented at SJCC, Boston, Mass., 1966 (unpublished).

25. T. O. Ellis and W. L. Sibley, "The Grail Project," *ibid.* (unpublished).

26. T. G. Stockham (personal communication).

27. E. E. Grant, "An Empirical Comparison of On-Line and Off-Line Debugging," SP-2441, Syst. Devel. Corp. (May 1966).

28. M. Halpern, "Computer Programming: The Debugging Epoch Opens," *Computers and Automation*, Nov. 1965.

THE SDS SIGMA 7: A REAL-TIME TIME-SHARING COMPUTER

Myron J. Mendelson and A. W. England

Scientific Data Systems, Santa Monica, California

INTRODUCTION

The SDS SIGMA 7 Computer system (Fig. 1) is unique among new computer designs in that it is the only system which has seriously considered and solved the problem of achieving true real-time response hardware and software capability while operating in a multiprogramming, multiprocessing, space-sharing, and time-sharing environment. This paper presents an overview of the system's architecture and describes in some detail those of its features which provide its unique capabilities.

The paper is divided into two major portions. The first part presents a succinct description of the architecture of the system. Its purpose is to acquaint the reader with the fundamental characteristics of SIGMA 7 and to provide a meaningful framework for the second section. The second section delineates seven major problems which were considered critical in the design of SIGMA 7 and presents the details of their solution.

DEFINITIONS

Multiusage

We will use the generic term "multiusage" to cover the spectrum of multiprogramming, multiprocessing, space-sharing, and time-sharing operations. These, together with the term "real time," will have the following meanings:

Real-Time Operation. A true real-time operation is one in which the response time requirements of the system are imposed by the time sensitive de-

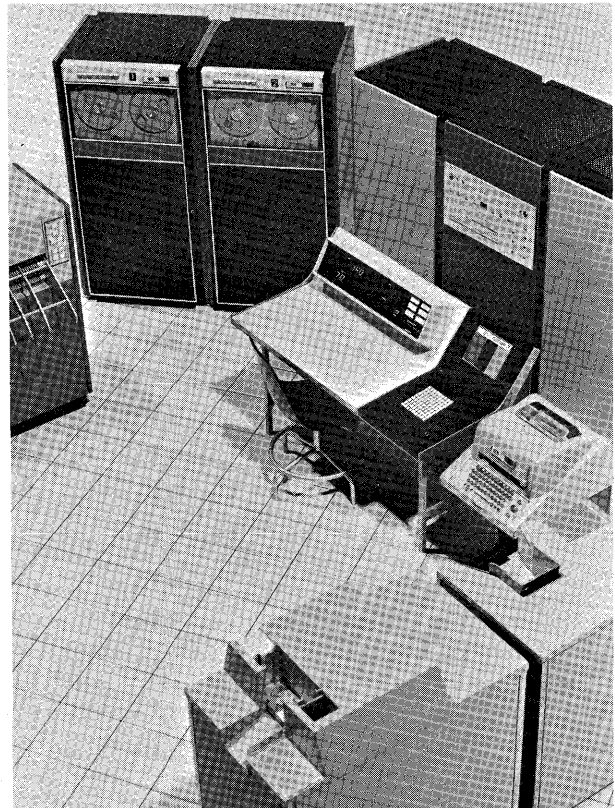


Figure 1. SIGMA 7 computer system.

mands of events external to the computer and its conventional peripheral equipment. Failure to meet this response time requirement results in true failure of the real-time system, not just degraded performance. The responsiveness of such a system is measured by the time interval between the arrival of an interrupt trigger signal and the execution of the first *useful* instruction in response to it. In the extreme case the maximum acceptable length for this interval may be measured in microseconds.

Multiprogramming. Multiprogramming is the concurrent operation of two or more independent programs in a single computing system, control being switched among programs through the actions of some central control program.

Multiprocessing. Multiprocessing is the simultaneous execution of one or more programs in a single computing system containing two or more processors, preferably sharing a common memory pool.

Space Sharing. Space sharing is the simultaneous residency in a common central memory of a number of independent (and perhaps concurrently operating) programs.

Time Sharing. Time sharing is a special case of multiprogramming in which a multiplicity of separate users have on-line, interactive use of a common system. It should be noted that neither multiprogramming nor time sharing imply space sharing but that space sharing is an essential ingredient in achieving true efficiency in these operations.

SYSTEM ORGANIZATION

Introduction

The following brief description of the SIGMA 7 system is presented in order to provide a meaningful framework within which to describe the specialized features which provide the system with a unique capability to meet its design goals. The SIGMA 7 is a modularly organized system which is configured out of a combination of Central Processing Units (CPU) (which contain Priority Interrupt Systems), Memory Modules, Fast Memory Units, Multiplexing Input/Output Processors (MIOP), Selector Input/Output Processors (SIOP), peripheral equipment Device Controllers (DC), peripheral Devices (D), and specialized real-time interfaces such as Analog-to-Digital Converters, Digital-to-Analog Converters, and Multiplexers (Fig. 2). This paper

will concentrate on the characteristics and structures of the CPU, IOP's and memory systems and their organization to meet the requirements of a broad range of operating environments.

Memory Organization

Core Memory Modules. The SIGMA 7 core memory is a 32 bit plus parity bit, word organized, 850 nanosecond cycle time unit which is available in module sizes of 4K, 8K, 12K, and 16K words ($K=1024$). The system architecture permits the inclusion and direct addressing of any size memory which can be configured within eight memory modules. This permits the structuring of 32 different memory sizes ranging from 4K words (16K bytes) to 128K words (512K bytes). Although the memory is word organized and word parity checked it is capable of altering less than a full word on a write operation. From 1 to 3 bytes may be written without altering the remaining bytes.

Multiple Ports. The processor/memory system complex is a bus-organized asynchronously operating system with each processor having its own private bus. The standard memory module is equipped with two independent access paths (called ports) and an optional third port may be added. Subsequent to the addition of a third port a memory port expander provides for the four way expansion of any *single* port so that a maximum of six independent buses may be connected to any memory module. The ports have a fixed priority relationship with respect to each other so that access request conflicts are automatically resolved.

Asynchronous, Overlapped Operation and Memory Interleaving. Memory operations may be initiated at any time and are not synchronized to any central clocking source. Memory operations are self-sustaining so that processor release occurs upon data acceptance (by the memory) on a write operation, and processor "go-ahead" occurs upon data availability on a read operation. This permits maximum utilization of CPU time and the overlapping of memory cycles with respect to a single processor or multiple processors in multiple-module memory configurations. To insure memory overlapping under any circumstances, address interleaving among several memory modules is provided on a two-way or four-way basis.

Fast Memory Units. An integrated circuit, non-destructively read fast memory unit with a read cy-

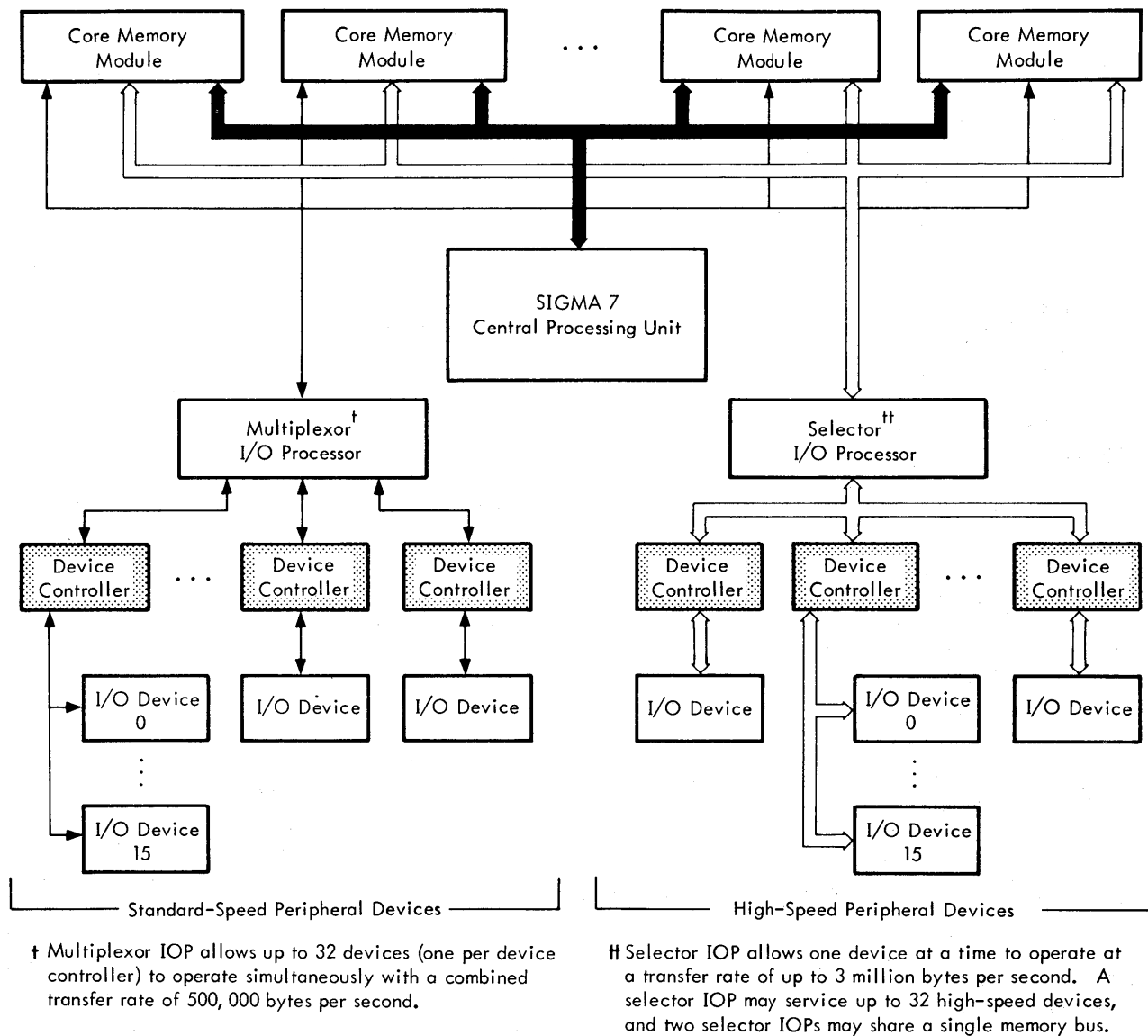


Figure 2. A typical SIGMA 7 system.

cle time of 60 nanoseconds and a write cycle time of 90 nanoseconds is used to implement a number of special functions within the SIGMA 7 system. The basic building block fast memory module provides 16 bytes of operating storage. Four such modules are combined to provide 16 words of scratchpad memory which serves as register storage for the CPU. Other combinations of this single module type are used for the implementation of memory protection systems, fragmentation and dynamic program relocation techniques, IOP channel control functions, and device buffering systems.

Every instruction makes one or more references to a set of sixteen registers. These registers are

stored in a sixteen word fast memory unit which is designated as a "register block."

In general, the SIGMA 7 register block can be used to provide:

1. 16 separate single precision arithmetic registers for fixed point word operations or short floating point operations.
2. 16 separate double precision arithmetic registers for fixed point half-word operations.
3. 8 separate double precision arithmetic registers for fixed point double precision operations or long floating point operations.

4. 7 separate index registers.
5. A decimal accumulator with a maximum capacity of 31 digits plus sign.
6. A significance position marking register for the EDIT instruction.
7. Control registers for byte string instruction implementation.

A unique design feature of the SIGMA 7 is that it may contain up to 32 blocks of registers. A 5-bit register pointer designates which of the 32 is currently active. The provision of multiple blocks makes it possible to preserve one register set and establish a new one within the 6 microsecond execution period of a single environment preserving and switching instruction.

Central Processing Unit

The CPU (Fig. 3) is a 32-bit, word-oriented, parallel-operating unit employing multiple registers in its instruction implementation. Its extensive instruction set provides for operations on 8 bit-bytes, 16-bit halfwords, 20-bit immediate operands, 32-bit words, and 64-bit doublewords.

Instruction Format. SIGMA 7 provides 106 major instructions, many of which have multiple modes of operation, all contained within a single instruction format. The Basic instruction is 32 bits in length and has the structure shown in Fig. 4. For a special class of immediate operand instructions the X and M fields are combined into a single 20-bit value which is sign extended and used immediately for computation with no further reference to memory for an operand.

Direct Memory Word Addressing. The 17-bit word address field in the primary instruction word permits the direct addressing of the maximum sized 128K word memory system. A memory address in the range 0-15 is used to designate the correspondingly numbered register and does not result in access to core memory. Hence, the full power of the instruction set may be applied to register-to-register operations as well as to register-and-memory operations.

Indirect Addressing. Indirect addressing is included for all instructions except those of the immediate operand class. If both indirect addressing and indexing are invoked, the indirect address operation is executed prior to the indexing operation.

Indexing. The indexing operation employed in

SIGMA 7 is unique. The indexing operation assumes that a list of either bytes, halfwords, words, or doublewords is stored beginning at the word address contained in the primary instruction word. If the designated index register is considered to contain the value K, the indexing operation, under control of the operation code (which establishes the operand length), produces the address of the byte, halfword, word, or doubleword displaced K units from this word location. Thus, the same index register may be used to locate the Kth operand of a list *independent of the operand length* (Fig. 5).

Instruction Set. The SIGMA 7 instruction set is comprehensive. It includes fixed point load, store, arithmetic, logical, and comparison operations for bytes, halfwords, 20-bit immediate operands, words, and doublewords. Optional floating point instructions provide full floating point arithmetic capability for both short (32-bit) and long (64-bit) formats. An optional set of decimal instructions includes full decimal arithmetic capability, plus Pack, Unpack, and Edit. Standard instructions are provided for manipulating byte strings up to 255 bytes in length. Single instructions are provided for moving a string, for comparing two strings, for the translation of a string from one character code to another, and for the scanning of a string for a specified set of characteristics. Push-down stack instructions provide for the efficient manipulation (including automatic stack limit checking) of arbitrary size stacks in core memory. Two generalized conversion instructions provide for the high speed conversion between any weighted binary information representation used external to the computer and its equivalent internal binary representation. Read Direct and Write Direct instructions provide for the direct communication between the CPU and external equipment without the use of an I/O channel. A comprehensive set of branch and system control instructions complete the instruction repertoire.

Typical instruction execution times (including indexing and mapping and excluding any memory overlap) are:

Fixed Point	
Add/Subtract	2.26 microseconds
Fixed Point	
Multiply	4.9 microseconds
Fixed Point Divide	12.5 microseconds
Floating Point	
Add/Subtract	
(short)	3.9 microseconds

Floating Point		
Add/Subtract		
(long)	4.5 microseconds	
Floating Point		
Multiply (short)	5.4 microseconds	

Floating Point		
Multiply (long)	8.0 microseconds	
Floating Point		
Divide (short)	12.3 microseconds	
Floating Point		
Divide (long)	24.5 microseconds	

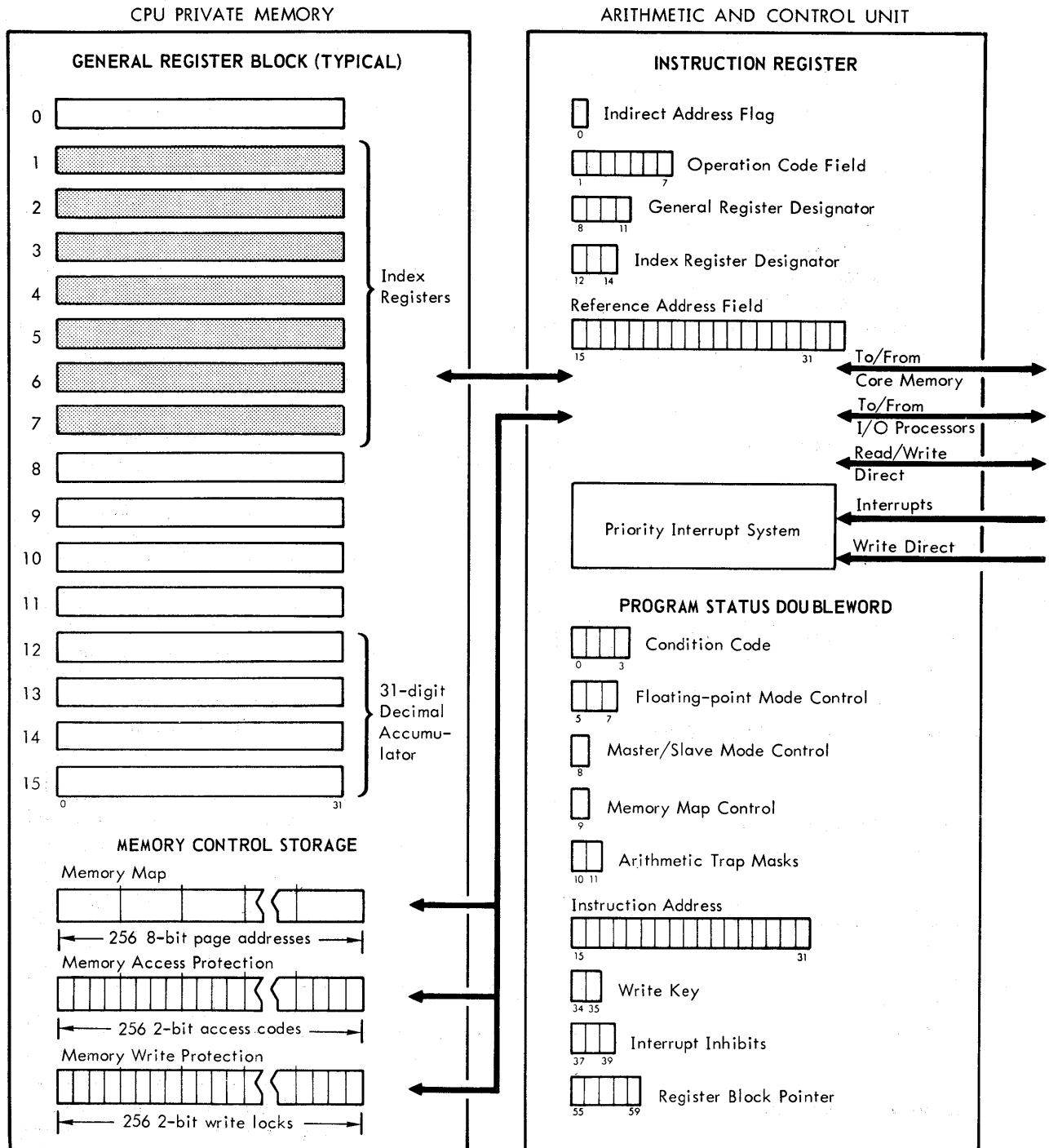


Figure 3. SIGMA 7 central processing unit.

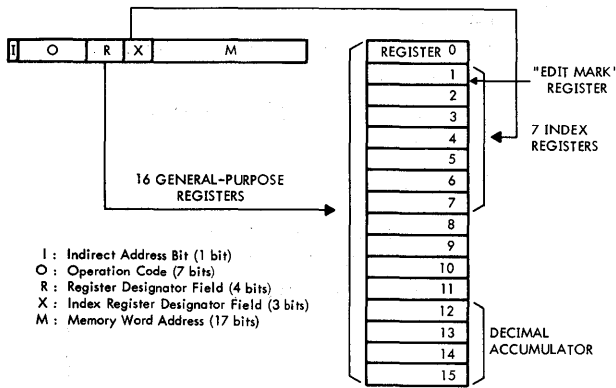


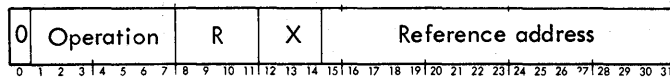
Figure 4. Register and instruction format.

Priority Interrupt System. The SIGMA 7 is equipped with the most powerful and flexible priority interrupt system currently available. Since this system constitutes one of the major elements contributing to the real-time responsiveness of the SIGMA 7 its description will be deferred to a later point in this paper.

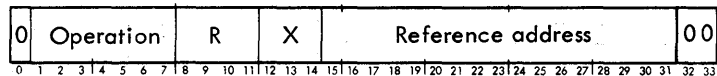
Input/Output Organization

Multiplexing and Selector Type Input/Output Processors. The Multiplexer Input/Output processor (MIOP) is designed to service a large number of slow to medium speed peripheral devices simultaneously. A single MIOP can provide concurrent service to as many as 32 devices having a total bandwidth of approximately 500,000 8-bit bytes per second. A single Selector Input/Output Processor (SIOP), with the capability of operating at rates up to 3 million bytes per second is designed to service any one of as many as 128 high speed devices which may be attached to it. SIOPs may have private buses or two may share a common bus. As many as eight IOP's may be attached to a single CPU. Each operates independently of the CPU under control of a stored program which is held in core memory. The CPU activates and monitors the I/O operations through the use of a set of five I/O instructions. Once activated the sequencing of the stored I/O

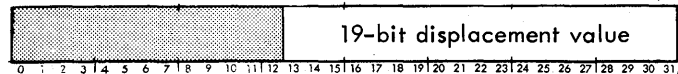
Instruction in memory:



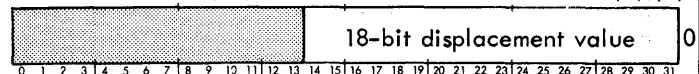
Instruction in instruction register:



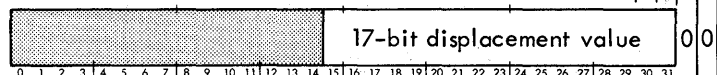
Byte operation indexing alignment:



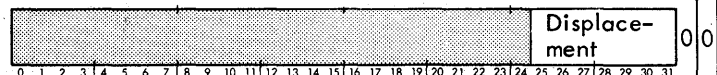
Halfword operation indexing alignment:



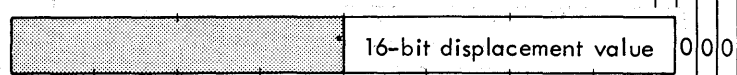
Word operation indexing alignment:



Shift operation indexing alignment:



Doubleword operation indexing alignment:



Effective virtual address:

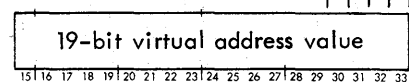


Figure 5. Index displacement alignment.

program is under control of the appropriate IOP with no further operations required by the CPU. CPU-I/O interaction is accomplished through I/O interrupts, the conditions for which are specified by the CPU in the I/O command list.

The IOP system operation is structured so that low cost, multiple channel, concurrent I/O operations which demand little CPU time for their execution are readily incorporated in the SIGMA 7 system.

Device Controllers and Devices. A wide range of 8-bit oriented peripheral equipment is available for attachment to IOPs. These include keyboard/printers high and low-speed paper tape input and output, punched card input and output, IBM compatible 7-channel multiple density magnetic tape units and single density 9-channel magnetic tape units, high speed line printers, fixed head rapid access disc storage units, communications equipment, and keyboard/display equipment. All such units are controlled by individualized Device Controllers which communicate with the IOPs through a common, simplified, electrical interface using a common method for control and information exchange.

Real-Time Interface Units. A full range of special systems equipment including such devices as Analog-to-Digital Converters, Digital-to-Analog Converters, and Multiplexers together with Device Controllers which interface them either with the direct input/output system of the CPU or with the standard IOP interface are also available.

SEVEN CRITICAL DESIGN PROBLEMS AND THEIR SOLUTION

General

This brief exposition of the SIGMA 7 system provides an over-all view of its principal features as a computing system, but it gives little insight into the special characteristics which uniquely permit it to carry out real-time tasks embedded in a multi-usage environment. Such an environment must be controlled by an executive program which allocates system resources; schedules operating intervals; provides services such as trap and interrupt response control, editing, compiling, assembling, and debugging; controls and executes I/O operations; swaps active programs between core and rapid access mass storage units; and guarantees the integrity, privacy,

and non-interference of all active programs and their associated data bases.

If a real-time operation is to be maintained in a multi-usage environment, it must have guaranteed dedication and protection of the system resources which it requires. Core and disk space must be assigned to it and protected from access by other programs. I/O channels, peripheral devices, and interrupt levels must be assigned, dedicated, and protected from outside interference. The establishment of a real-time operation and the dedication of resources to it should be dynamically available through the operating system. These tasks must be accomplished in such a way as to permit full freedom and capability to the non-real-time operations while in no way degrading the responsiveness of the system to the time-sensitive demands of the real-time program. In the following section we will describe the design problems which were faced in meeting these requirements and present the SIGMA 7 structures which provide for their solution.

The Problem of Priority Interrupts

The system must be equipped with a true priority interrupt system which is flexibly structured and controlled and whose operation in establishing priorities and recording and sequencing interrupt requests is essentially instantaneous and independent of CPU action. Interrupts of higher priority must be permitted to interrupt partially completed responses to those of lower priority. To maintain fast response, interrupt requests should require no decoding action on the part of the CPU to determine their source or nature. Capability for dynamically varying the priority sequence to meet the demands of a changing environment must be available. *No other system element may be designed such that its proper operation requires the inhibition of the priority interrupt system for any period of time.*

The SIGMA 7 Priority Interrupt System. The SIGMA 7 interrupt system is best described from the ground up. The basic interrupt level has four mutually exclusive states which are designated as Disarmed, Armed, Waiting, and Active. A separate flip-flop is used to disable or enable the level (Fig. 6).

In the Disarmed state the interrupt level rejects all incoming interrupt trigger signals. In the Armed state the interrupt level will accept a trigger signal from an outside source, or from the CPU, and will

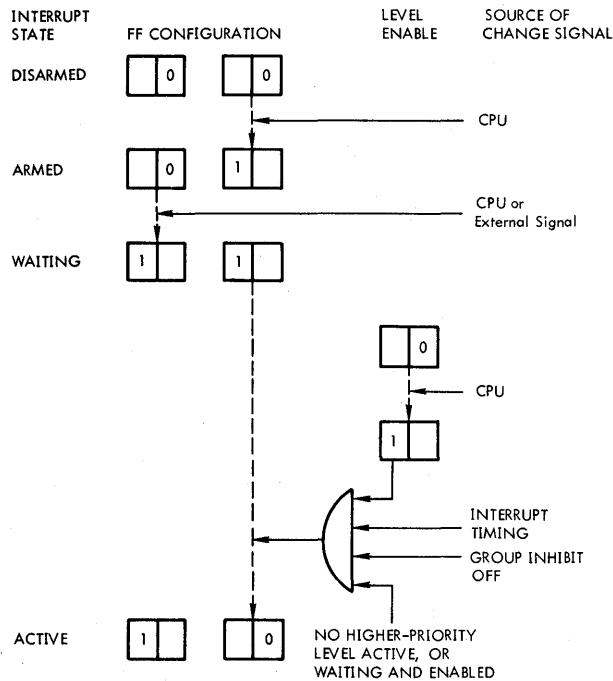


Figure 6. Interrupt level operations.

move to the Waiting state, where it will remain until the level is acknowledged by the CPU. If the level is disabled any Waiting condition is held in abeyance, preventing it from entering the priority chain of requests for CPU action. All Enabled and Waiting interrupt levels are permitted to enter the priority chain of requests awaiting computer interrupt response action.

Interrupt levels are organized into four classes which are designated as the Over-ride Class, the Counter Class, the I/O Class, and the External Class. The Over-ride class can never be Inhibited, Disarmed, or Disabled. A separate inhibit flip-flop is provided in the CPU for each of the other three classes, so that the CPU can prevent an entire class from entering the priority request queue. In effect this inhibit flip-flop disables the class regardless of the Enable-Disable states of the individual levels within it. The External Class is further divided into 14 groups each containing 16 interrupt levels. The priority request queue starts at the Over-ride Class and then may be threaded through the remaining Classes (and Groups of the External Class) in any order which the customer may desire. Thus, external interrupts may be given priority positions above, below, or in between those allocated to the Counter Class and the I/O Class (Fig. 7).

Each interrupt level has a unique location in low order memory dedicated to it. Control of the CPU is automatically forced to this location when the interrupt is acknowledged and permitted to move to the Active state. This action occurs whenever the highest priority Waiting, Enabled, and Uninhibited interrupt level is of higher priority than the highest priority currently Active interrupt level.

The CPU can control the states of the interrupt system. A group of sixteen interrupt levels are operated upon simultaneously under control of a sixteen bit mask which selects the subset of the sixteen to be modified. Operations which may be performed upon the mask-selected levels include Disarm, Arm and Enable, Arm and Disable, Enable, Disable, Load Enables, and Trigger. The Trigger function permits the CPU to apply an interrupt signal to its own interrupt system. This feature can be used to simulate an external interrupt environment for purpose of system checkout. It also permits the CPU to carry out the highly time sensitive portion of an interrupt response and then to create for itself a low priority interrupt to call for the deferred servicing of the less time sensitive portion at a less pressing time.

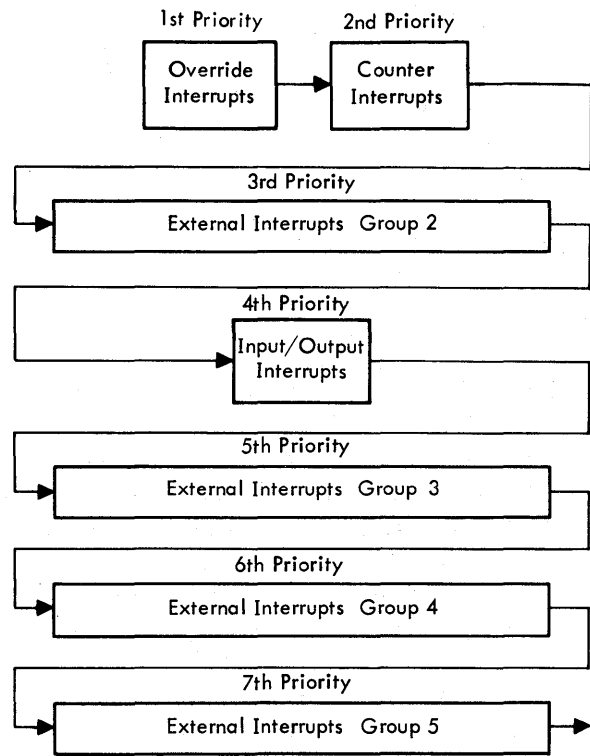


Figure 7. Typical interrupt priority chain.

The Problem of the Duration of Uninterruptible Intervals

Such an interrupt system is of little value if the CPU can remain for any significant period of time in an uninterruptible state. Under normal operating conditions, the longest uninterruptible interval must be kept short, and under abnormal conditions no malfunctioning peripheral hardware or software may be allowed to "hang up" the CPU in a noninterruptible state.

SIGMA 7 Interruptible Instructions and the Watchdog Timer. To insure that the longest uninterruptible interval which the CPU may experience in normal operation is short, all long instructions have been designed so that they may be interrupted *during* the course of their execution. Registers are held in fast memory, but instruction execution occurs in hardware elements. Since the original operands are retained in fast storage until instruction execution is completed, instruction aborting occurs without loss of information. Instructions whose duration is less than 10 microseconds are never aborted. Instructions in the 10-30 microsecond range are designed so that they may be aborted and subsequently restarted upon return from the interrupt. Instructions whose execution time exceeds 30 microseconds are designed so that they may be aborted and subsequently have their execution resumed from their point of interruption upon return from an interrupt process.

An instruction "watchdog timer," included in the standard SIGMA 7 configuration, guarantees against hardware hang-up by insuring that the time interval between interruptible points never exceeds 40 microseconds.

The Problem of Red Tape Time

Mere capability to initiate action in response to an interrupt is of little use to a real-time situation if it requires an inordinate amount of time to preserve the operating environment which exists at the time of the interrupt and to establish the new environment required for the processing of the interrupt. Hence, an extremely rapid context preservation and switching system must be provided in order to assure that minimum time lapse exists from the initiation of interrupt response to the execution of operations which are truly pertinent to the demands of the interrupt situation. Such a switching system must be repeatable to any number of levels in order to accommodate interrupts of interrupts.

SIGMA 7 Context Switching. A single instruction, Exchange Program Status Doubleword (XPSD) results in the collection of all of the active control states of the CPU and their storage in an arbitrarily designated doubleword location in core memory. This instruction execution then proceeds by loading the active control states with correspondingly structured information contained in the following two words in memory. Thus, the entire control environment of the CPU is stored and reloaded in six microseconds with the execution of a single instruction. A return to a prior control state is accomplished through the execution of another single instruction, Load Program Status Doubleword (LPSD), which also provides for clearing and arming or disarming the highest level active interrupt. An XPSD at the interrupt location saves the old environment and establishes the new one for the interrupt response. An LPSD at the conclusion of the interrupt process returns the CPU to its state prior to the interrupt. Since the storage and access locations designated by the XPSD and LPSD instructions are arbitrarily located in memory, nested chains of interrupted interrupt routines may occur to any level without loss of control and with automatic denesting as interrupt processes complete.

A second major element of context saving is the preservation of register states. Registers may be preserved in memory and restored through the use of multiple register load and store instructions or may be preserved in core implemented stacks through the use of multiple register push and pull instructions. Even the high speeds of these operations may result in too great an overhead time for some real-time processes; hence, a register storing and loading technique which is accomplished during the execution time of an XPSD instruction is provided. This technique is available whenever the CPU is equipped with one or more of the optional additional register blocks. The 5-bit Register Pointer is a portion of the contents of the Program Status Doubleword which is stored and loaded with the XPSD instruction. Hence, if a register block is available and dedicated to a real-time process the execution of the XPSD instruction which initiates the process automatically preserves the control context and the register context of the interrupted routine and automatically establishes the corresponding contexts for the interrupt process. Under these circumstances, the equivalents of register preservation, loading, and restoring are all accomplished within the execution

times of the XPSD and LPSD instructions which initiate and terminate the interrupt routine (Fig. 8).

The Problem of System Integrity

Some means must be provided to guarantee the integrity of the executive system and for it, in turn, to establish and guarantee the integrity of all other programs.

Master/Slave States and Privileged Operations.

The SIGMA 7 CPU can operate in either a Master or Slave state. In the Master state all instructions can be executed normally. In the Slave state instructions whose execution are critical to the integrity of system resources are illegal. Such instructions are designated as Privileged Operations and are reserved to programs operating in the Master state. Privileged operations include all instructions which affect Input/Output operations through the Input/Output system, Input/Output operations direct to memory, the memory protection systems, the interrupt system, the operating state of the CPU (e.g. a Slave state program cannot switch itself to the Master state), or the continuation of system operation.

Memory protection, the other aspect of guaranteeing system integrity, is presented in the following section.

The Problem of Space Sharing

Efficiency in multiusage implies the simultaneous residency of many programs, or portions of programs, so that when conditions require that control be given to a new program it is resident and such action can occur immediately. Thus, under control of the executive system, partially executed programs must be permitted either to space share or to be swapped out of memory and later returned, preferably to whatever space is available. When a program is held up for I/O operations, only its I/O buffer region should be retained in core with the remainder of the program dumped to disk so that its space in core may be available for other usage. As such actions take place, the available memory space rapidly becomes fragmented into discontinuous regions which should be directly usable without having to repack the memory in order to achieve contiguity. Thus, a system should be provided for the execution of programs which have been dynamically relocated into discontinuous memory regions.

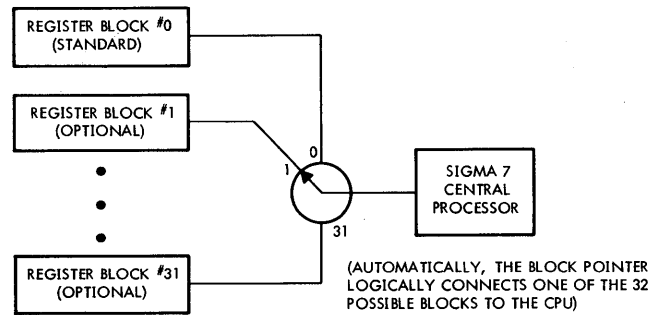


Figure 8. Block pointer and register selection shown with a block pointer value of 1 (00001).

The SIGMA 7 Memory Map. Dynamic program relocation into discontinuous fragments of memory is provided through the incorporation of an optional feature, the memory map. If the map option is installed, any program may be broken into 512-word pages and distributed throughout the implemented core memory in whatever 512-word pages of space are available. The memory map then permits the program to be executed as though it were located in the contiguous region of memory for which its addresses have been established. Clearly, the map provides the transformation of Virtual Addresses (i.e., addresses generated within a program such as instruction addresses, operand addresses, and indirect addresses) into Real Addresses (i.e., the physical core addresses where program-designated values are actually located).

The memory map employs a 256-byte, integrated circuit memory in its implementation, and thus provides for the mapping of a full 128K Virtual Address space. A mode flip-flop designates whether a program is to operate in a mapping or non-mapping mode. When mapping is invoked, the following events occur every time an actual reference to memory is to be made (Fig. 9):

1. The 17-bit address generated by the program is broken down into a 9-bit word address and an 8-bit page address.
2. The 8-bit page address is used to access one of the 256-byte map memory locations.
3. The 8-bit page address stored at that location replaces the 8-bit page address portion of the Virtual Address to form a Real Address.
4. The Real Address is used to access the memory.

Because of the speed of the integrated circuit memory, these actions add only 60 nanoseconds to each memory access.

A special instruction, Move to Memory Control, provides for rapid changing of the memory map.

With the map option, a program can be brought in and distributed to any set of 512-word pages which may be available in memory. The Move to Memory Control instruction is then used to write the map so that the proper address transformation will be made. The mapping mode is then entered and control is turned over to the program, which

then operates as though it were located in the contiguous region of memory for which it was designed. The operation of such a program may be halted at any time, the program subsequently relocated to any other set of 512-word pages, the map rewritten, and the program operation resumed with no adverse effects.

Programs whose addresses range over the 128K Virtual Address space may be executed on a machine with far less than 128K words of implemented core. The map permits portions of such programs to be resident and operate in available core space. Pro-

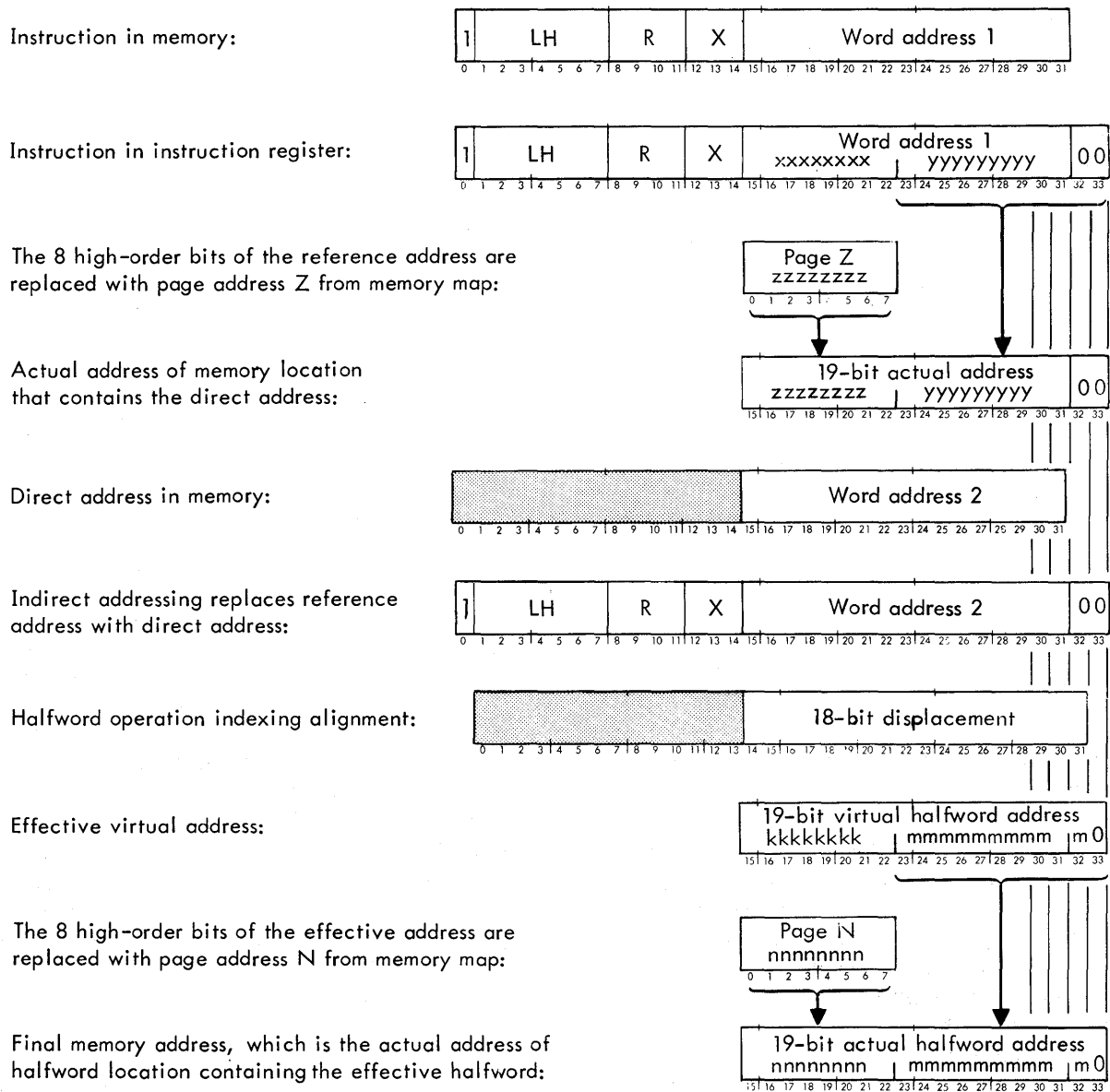


Figure 9. Example of generation of actual memory addresses; indirectly addressed, halfword operation.

gram references to blocks which are not resident are automatically trapped so that a page-turning system may be readily implemented.

A number of design compromises were made in the incorporation of the map in SIGMA. The most important of these was the decision not to incorporate a two-level (segment and page) mapping structure. Consequently, all programs which must directly communicate with each other, without the intervention of the executive system, must share a common Virtual Address space since they must share a common map. This includes the executive system itself which must provide services to user programs. When doing so, the executive system must operate in the mapping mode since no unique bit was available in each instruction word to designate whether or not to employ mapping on an individual instruction execution basis. Thus, in the interests of simplicity and limitation of costs, the map system has been deliberately incorporated in such a way that a user's Virtual Address space is curtailed by the size of the executive system and the public routines and services to which the user's program desires to have access. Further, these latter programs must have dedicated space in the Virtual Address space of all users who desire to use them so that they may maintain constant residency in all users' maps. While these limitations were recognized it was felt that it was worth far more to achieve the powers of the mapping operation at a price which would bring them to a large segment of the market, than it was to achieve full segmentation for a much smaller portion of the market.

The Problem of Memory Protection

An additional aspect of guaranteeing the integrity, privacy and non-interference of all active programs is that of memory protection. Early implementations of memory protection were aimed almost exclusively at providing the write protection function which is essential for guaranteeing that one program cannot destroy another. The multi-usage environment adds further dimensions to memory protection requirements. Privacy considerations of privileged information (such as payroll data) require that portions of memory be protected from unauthorized reading as well as writing. The complexity of the operating environment makes it highly desirable to catch errant programs at the earliest possible time. This desire leads to the concept of instruction protection which prevents a program from executing an instruction

taken from an instruction-protected region of memory.

Access Protection. An additional 512 bits of fast memory are supplied with the map option. These provide storage for two Access Protection bits which are associated with each of the 256 Virtual Address pages. These bits are accessed during the mapping operation whenever the CPU is in the Slave state. They are used to impose inhibitions on a slave program's use of the information in the page which it is attempting to access. These inhibitions are designated in the following table:

Access Protection Value	Inhibition/Permission Control
00	Permit slave access to this page for any purpose
01	Inhibit slave access for writing, but permit instruction or operand read access
10	Inhibit slave access for writing and instruction execution, but permit operand read access
11	Inhibit slave access for any purpose

Note that these inhibitions are imposed on slave Virtual Addresses and are in effect no matter where the slave program may be located physically in core.

The Access Protection bits are used to restrict the operation of a slave program to its allocated addressing domain, and within that domain to permit the establishment of read-only or read-and-execute only pages of information. Thus, provision is made to guarantee secrecy and preservation of sensitive information, for common use of non-destructible data bases and public subroutines, and for the trapping of run-away program attempts to execute data.

The 64-byte Access Protection fast storage area is loaded with a Move to Memory Control instruction. Because of the existence of this form of program inhibition, the memory map need only be loaded for the address domain over which a slave program is expected to operate. The Access Protection bits are loaded for the full 128K Virtual Address domain and thus are guaranteed to protect against slave program operations in pages outside their prescribed domain. This fact reduces overhead time involved in map loading for slave programs with restricted addressing ranges.

Memory Write Protection. The Access Protection bits operate over the Virtual Address domain, are effective only for slave programs, and are not available unless the Memory Map option is installed. Consequently, an optional memory write protection feature which operates independent of the Access Protection bits is also available. The memory write protection feature operates in both the Master and Slave states. This feature is implemented with a 512-bit fast memory unit which stores a 2-bit write protection "lock" for each 512-word page. Every operating program is given a 2-bit "key" which, in conjunction with the locks, controls its write access to a page in the memory according to the following rules:

1. If the lock value for the page is 00, writing is unconditionally permitted. That is, the page is "unlocked."
2. If the key value for the program is 00, writing is unconditionally permitted. That is, the program has been given a "skeleton key."
3. If the lock and key values are both non-zero, then writing is permitted if, and only if, the lock and key values are identical.

Note that this feature is associated with the use of Real Addresses and, therefore, supplies write protection for physical memory. If the map option is installed, both forms of memory protection are operative, the Access Protection bits operating on the Virtual Address space of the program and the locks and keys on the physical memory space, after mapping. Locks are installed through the use of the Move to Memory Control instruction. The memory write protection system makes it possible to provide memory protection in the absence of the Memory Map. It also provides memory protection for simultaneously resident Master mode programs, thereby guaranteeing their integrity and the integrity of publicly available, reentrant, pure procedures which service users of both classes. This form of memory protection also provides a powerful tool for the development or revision of portions of the executive system. Such a development can occur on-line, while the system is operating, since the unchecked portion can operate under a write protection constraint which guarantees the memory integrity of the system.

The Problem of Recursive and Reentrant Routines

Efficient operation in a multi-usage environment requires efficient utilization of memory and minimization of program swapping time. The provision of single, public copies of routines which are used in common by many concurrently operating programs is an essential ingredient in optimizing both of these functions. Public routines avoid multiple copies, one for each user, and eliminate the swapping time associated with their transmission between core and rapid access disk. (Indeed, swapping out time is always avoided for all pure procedures.) Public routines must be pure procedures which operate on a designated context. When the context and working space are provided by the calling program and several such programs may be concurrently using the routine it is said to be reentrant. When such a routine may repeatedly call itself, and, therefore, be required to provide its own nested context and working space, it is said to be recursive. A single routine may be both recursive, i.e., capable of calling itself, and reentrant, i.e., capable of being called by many different programs prior to its completion of operations for any single one of them. Of primary importance is the requirement that public routines must operate in an interrupting environment in which they may be invoked by one or more programs before completing their operations for another. The primary SIGMA 7 design constraint that no software may be designed so that it must turn off the interrupt system in order to be guaranteed to operate properly is particularly difficult to meet in this environment. These requirements place demands on the hardware to provide entry and context establishing methods which provide for efficient and dynamic utilization of space and guard against loss of information or control under all operating conditions.

SIGMA 7 Hardware Features for Reentrance and Recursion. Both reentrance and recursion require an efficient means for guaranteed preservation of the context of a partially completed process, including the 16 general registers and the link address, and for the institution of the corresponding context for a new user. A Branch and Link instruction which preserves the program address in a designated register provides a simple and effective linking mechanism for both reentrant and recursive routines. The indirect addressing and indexing mechanisms provide one of the means for context designation in the reentrant case. Multiple register blocks provide a rapid means

for context switching. The memory map provides the most direct and effective method for both context designation and context switching since it permits the reentrant routine to directly address its designated context. Switching from context to context then merely requires a map change.

Both types of programs require register preservation. The Load Multiple and Store Multiple instructions provide a ready solution for the reentrant case since the calling programs must provide storage space within their context regions. The recursive case is more complex since the recursive program must provide its own storage. In this case, however, such storage is always used in a nested fashion on a last-in-first-out basis. The SIGMA 7 pushdown stack manipulating instructions provide the ideal solution for this situation. There are five instructions in this set, PUSH, PULL, PUSH MULTIPLE, PULL MULTIPLE, and MODIFY STACK POINTER. These stack manipulating instructions provide an efficient means for moving information between single or multiple registers and core locations which are contained within a pushdown stack which is under control of a doubleword stack pointer (Fig. 10).

The stack pointer contains the address of the top of the stack, a count of the words in the stack, a count of the number of spaces currently available in the stack, and stack underflow and overflow trap mask bits. With such a mechanism, recursive entry to a routine merely requires the execution of a single PUSH MULTIPLE instruction to preserve the current context in a stack for which space is provided within the routine itself. A routine which is both reentrant and recursive merely uses pushdown stacks which are stored within the context regions of the various calling programs.

In general, the pushdown stack mechanism provides a powerful tool for any dynamic space allocation situation in which a last-in-first-out nesting of information is guaranteed.

Other Multiusage Features. The limitations of space do not permit the description of many of the details of the SIGMA 7 which make for efficiency of operation in a multiusage environment. A few additional features are worthy of mention, however. A

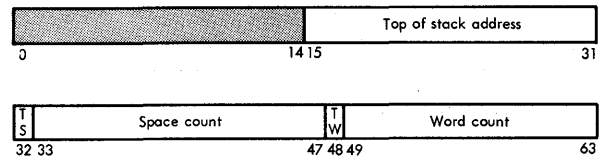


Figure 10. Stack pointer.

set of four Call instructions, each providing sixteen independent branches, generate a total of 64 generalized operator or subsystem entrances. The Call, operating through the Sigma Trap system and the use of XPSD instructions, provides a mechanism for accessing generalized, re-entrant service routines. The Call mechanism provides the proper control states for these routines and establishes the means for returning to the control state of the calling routine, without going through the executive program and without using the address portion of the Call instruction itself. Consequently, the address portion of the Call instruction is available for the designation of operand(s) to the called routine. Calls thus can be considered a generalization of the SDS Programmed Operator concept.

The SIGMA 7 CPU is equipped with two real-time counters, and 60 cycle, 1KC, 2KC, 4KC, and 8KC clock sources, any of which may serve as inputs to the counters. Optionally, it may be equipped with two additional counters. Clock pulse counting is handled by single instruction interrupts which cause counts to be accumulated in arbitrarily designated memory cells. Overflows from these locations cause a second interrupt, unique to each counter, to occur. Hence, real-time synchronization may be maintained, elapsed time intervals may be measured, and arbitrary length count down timers may be established, all without recourse to elaborate, software-derived timing routines.

An extensive error-trapping system provides for automatic error detection and recovery from situations which would otherwise eliminate all possibility of interrupt responsiveness.

An optional power fail-safe system provides for the detection of incipient power failure and the orderly shut-down of the system so as to preserve its operating state. An automatic start-up procedure is initiated upon restoration of power.

TECHNOLOGICAL FOUNDATIONS AND FUTURE DIRECTIONS OF LARGE-SCALE INTEGRATED ELECTRONICS

Richard L. Petritz

*Texas Instruments Incorporated
Dallas, Texas*

INTRODUCTION

The technological base of the electronics industry has undergone dramatic change in the past 20 years, largely related to the expansion of the use of materials technology. With the invention of the transistor in 1948, semiconductor materials processing provided the technology for an entirely new class of electronic devices. The invention of the monolithic integrated circuit in 1958 extended the use of materials technology to the formation of complete circuit functions on chips of semiconductor. We are now entering another phase of the expansion of materials technology, in which complete equipment components will be processed on slices of semiconductor.

It is the purpose of this paper to discuss the technological foundations and future directions of this latter phase.

This phase has already been given several names, some of which are "large-scale integration" (LSI), "computer on a slice," and "array technology." The term "large-scale integration" is close to being the most descriptive, although at times the syntax is awkward. A somewhat more precise term is "large-scale integrated electronics." We will use LSI to abbreviate both "large-scale integrated electronics" and "large-scale integration."

The products of large scale integrated electronics will be called Integrated Equipment Components (IEC's) to distinguish from integration to the circuit function (Integrated Circuits, IC's).

In order to set the stage for the discussion of large-scale integrated electronics technology, Fig. 1 is included for review of the technologies of discrete semiconductor devices and monolithic integrated circuits. The reader is referred to the December 1964 issue of *Proceedings of IEEE*¹ for a comprehensive review of integrated electronics. The article by Jay Lathrop² is an excellent discussion of integrated circuits technology. A discussion of the history of semiconductor technology is contained in Ref. 3, and the status of large-scale integration technology in 1965 is reviewed in Ref. 4. Recent published reports of meetings concerning LSI are listed in Ref. 5.

An important conclusion of Fig. 1 is that there are potentially 40,000 gates per 1-inch slice of silicon. The use of a 1" slice of silicon is arbitrary—the industry is moving to larger slices. Today 1¼" is widely used and 3" diameter is forecast by 1976. A principal goal of LSI technology will be to utilize this logical power in slice form; that is, interconnect the gates such that powerful logic and memory functions are formed on single slices of semiconductor material.

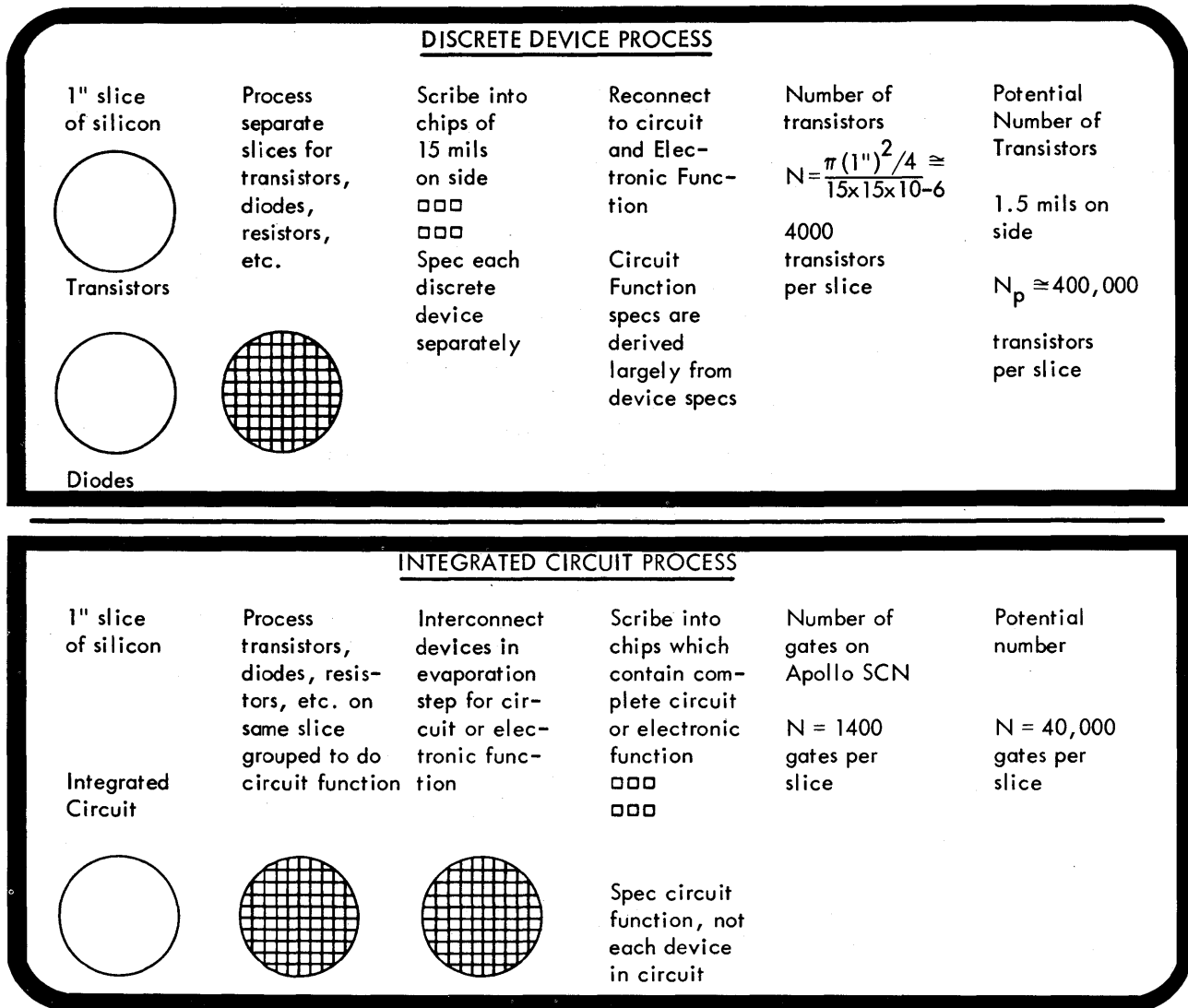


Figure 1. Discrete device and integrated circuit processes.

Interconnections Required to Construct Equipment Components

Before going further in our discussion of the technological foundations of large-scale integrated electronics, let us discuss the problem of building an equipment component, such as a memory or central processor unit, requiring the logical power of, for example, 10,000 gates.

In Fig. 2 it is shown that the construction of an equipment component of 10 K gates from discrete devices requires about 150 K mechanical connections. We have assumed each gate has 10 internal and 5 external connections. Since each gate is connected to something else, a linear slope brings us to 150 K for an equipment component of logical power of 10 K gates.

For multifunction integrated circuits, the 10 mechanical connections needed to interconnect the discrete devices to form gates are made by materials processing (evaporation of metal). Thus a single gate has five terminal connections, and 50 K mechanical connections must be made to achieve an equipment component complexity of 10 K gates (Fig. 2).

We thus conclude that while integration to the circuit function level of complexity has reduced the mechanical interconnection problem (from 150 K to 50 K in this example), there are still a great many mechanical interconnections to be made in order to construct an equipment component.

It is of interest to observe that the use of equipment components as black boxes requires orders of magnitude less external connections, that is, most

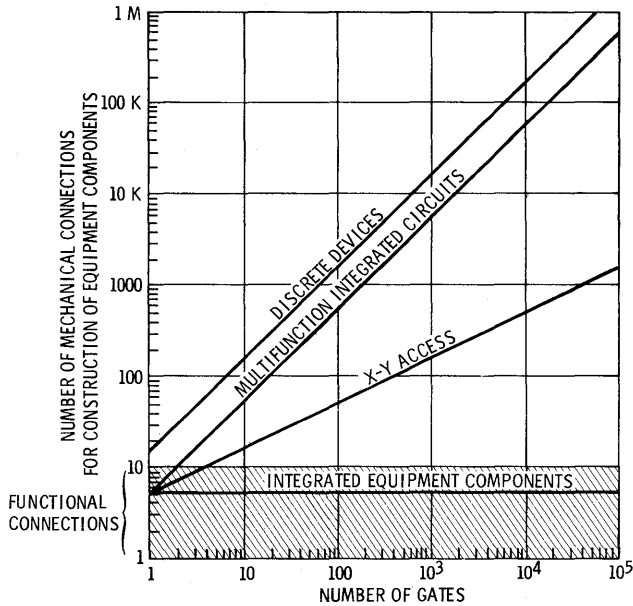


Figure 2. Number of mechanical connections for equipment components vs number of gates.

of the 50 K connections are internal. In Fig. 3, bottom left hand corner, it is shown that a typical ALU of a computer requires 85 external connections to add two 25-bit words.

Defining R as the number of functional connections per bit of processed data,

$$R = \frac{85}{25} = 3.4$$

We note that the other equipment components shown in Fig. 3 have R values in the range of 1–10, relatively independent of the number of gates per equipment component. The shaded area on Fig. 2 will contain the number of functional connections per bit for equipment components. Comparison of the shaded region with the lines for discrete devices and multifunction integrated circuits shows that most of the mechanical connections are internal. Essentially all connections above the shaded area are internal.

As will be developed in detail below, a principal goal of large-scale integration technology is to make these internal connections a part of the materials processing technology (e.g., by evaporation).

Those equipment components in which most of the internal connections are made by materials processing technology we shall call Integrated Equipment Components (IEC's), to distinguish them from Integrated Circuits (IC's) and from equipment components fabricated principally by mechanical tech-

niques. Figure 4 illustrates these definitions qualitatively.

Technologies Suitable for Integrated Equipment Components

The essential requirements of a basic technology are two:

1. It must be capable of forming thousands or more of both active and passive devices in or on a common substrate.
2. It must be capable of interconnecting thousands of active and passive devices into IEC's by a materials process such as evaporation without separate mechanical handling of devices.

Clearly monolithic semiconductor integrated circuit technology has the potential of meeting both of these requirements as shown in Figs. 1 and 4. This paper will be concerned principally with semiconductor technology, but before developing this further, let us consider what other technologies may also be suitable for large-scale integrated electronics. The consideration that the technology must be capable of forming complete electronic functions without intermediate steps of dicing and mechanical handling eliminates the hybrid technologies as we know them today (e.g., thick or thin films, with chip transistors). However, hybrid and discrete device technology, in combination with monolithic semiconductor technology, will add to the flexibility of LSI.

Technologies in addition to silicon that appear to have promise include: thin films, where the TFT

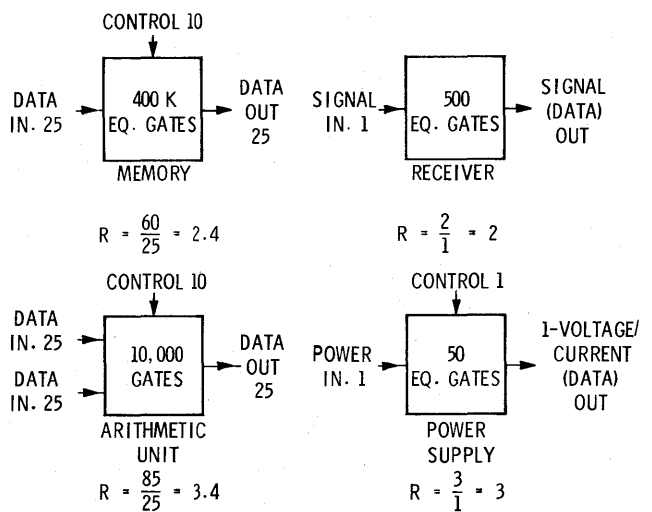


Figure 3. Ratio of functional connections to bits of output data for equipment components.

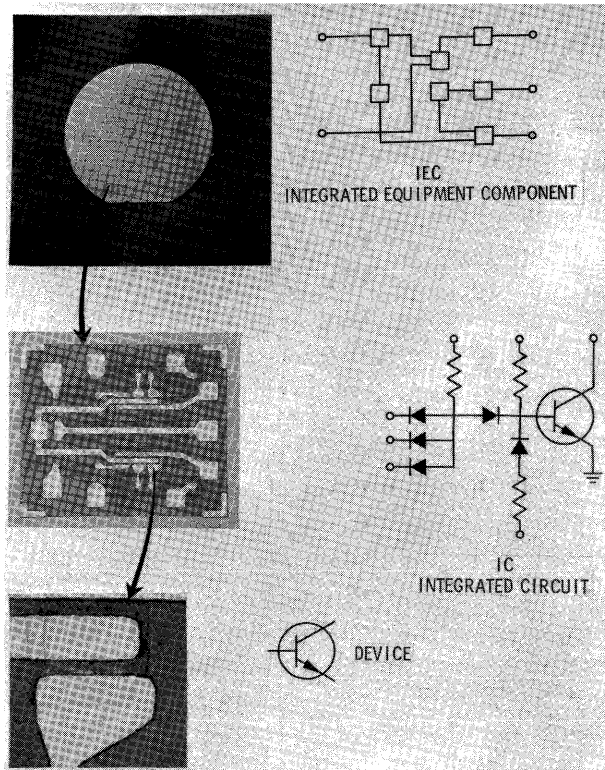


Figure 4. Pictorial view of integrated equipment component (IEC), integrated circuit (IC), and semiconductor device.

(thin film transistor) now offers a method for achieving an active element as an integral part of thin film technology; cryoelectronics, where logic, control, and memory functions are processed in compatible steps; and compound-materials technology, where more than one material may be involved in the processing.

Applicability of LSI Technology to Linear as Well as Digital Functions

It is clear that LSI technology is directly applicable to digital functions; the impact on linear functions is not so clear. A chief characteristic of digital functions is that signals are propagated through many levels of logic with no basic change in the signal level. The power gain supplied by the active element compensates for losses in the system. However, the signal level stays at the same amplitude (generally about 1 V for silicon circuits). Thus, long chains of logic functions can connect together with a high degree of repeatability of the basic logic circuit (normally a NAND or NOR gate). Consequently, digital functions lend themselves rather naturally to LSI technology.

Linear functions are generally concerned with changing the level of the signal function. For example, in a low-noise amplifier the signal level is amplified from a few microvolts to a level of a few volts. Since there is a finite lower level (set by noise) and a finite upper level (set by the particular application) over which amplification must occur, a finite number of stages is required for linear applications. An initial judgment is that array technology will not have a major impact on linear functions.

However, one must temper this conclusion because there is a large class of linear functions that requires a number of parallel channels, each identical. Consider as an example a solid-state replacement for a vidicon. With an array of photodetectors, it would be desirable to place a linear amplifier at each detection point in order to build up the signal level before it is multiplexed into a single channel. Thus, an array of photodetectors, combined with linear amplifiers, is an example of array technology impacting linear functions.⁶ Arrays of compound semiconductors, consisting of photodetectors and linear amplifiers, offer a possibility of vidicon-like sensors operating in the infrared spectrum. There are other examples, including sense amplifiers for memories and light displays.

General Aspects of Large-Scale Integration Technology in Semiconductors

The calculations of Fig. 1 suggest that silicon slice processing has the potential of fabricating complete equipment components (IEC's) on a slice of silicon. Let us now outline the basic technological approaches being pursued for accomplishing this end. With the semiconductor approach, two broad areas of effort can be identified as discussed below and shown in Fig. 5 and 6.

Device-Based Design. The left side of Fig. 5 shows the approach which seeks to achieve complete equipment components by incorporating a relatively large number of devices within an area of silicon. This type of IEC is designed directly from devices and no particular effort is made to define unit circuits. The important distinction between this approach and the multifunction integrated circuit is that IEC's are achieved by incorporating many more interconnections on the chip. A typical example of an IEC made by this approach is a 50-bit MOS shift register.

Circuit-Based Design. The second broad approach to large-scale integrated electronics is shown schemati-

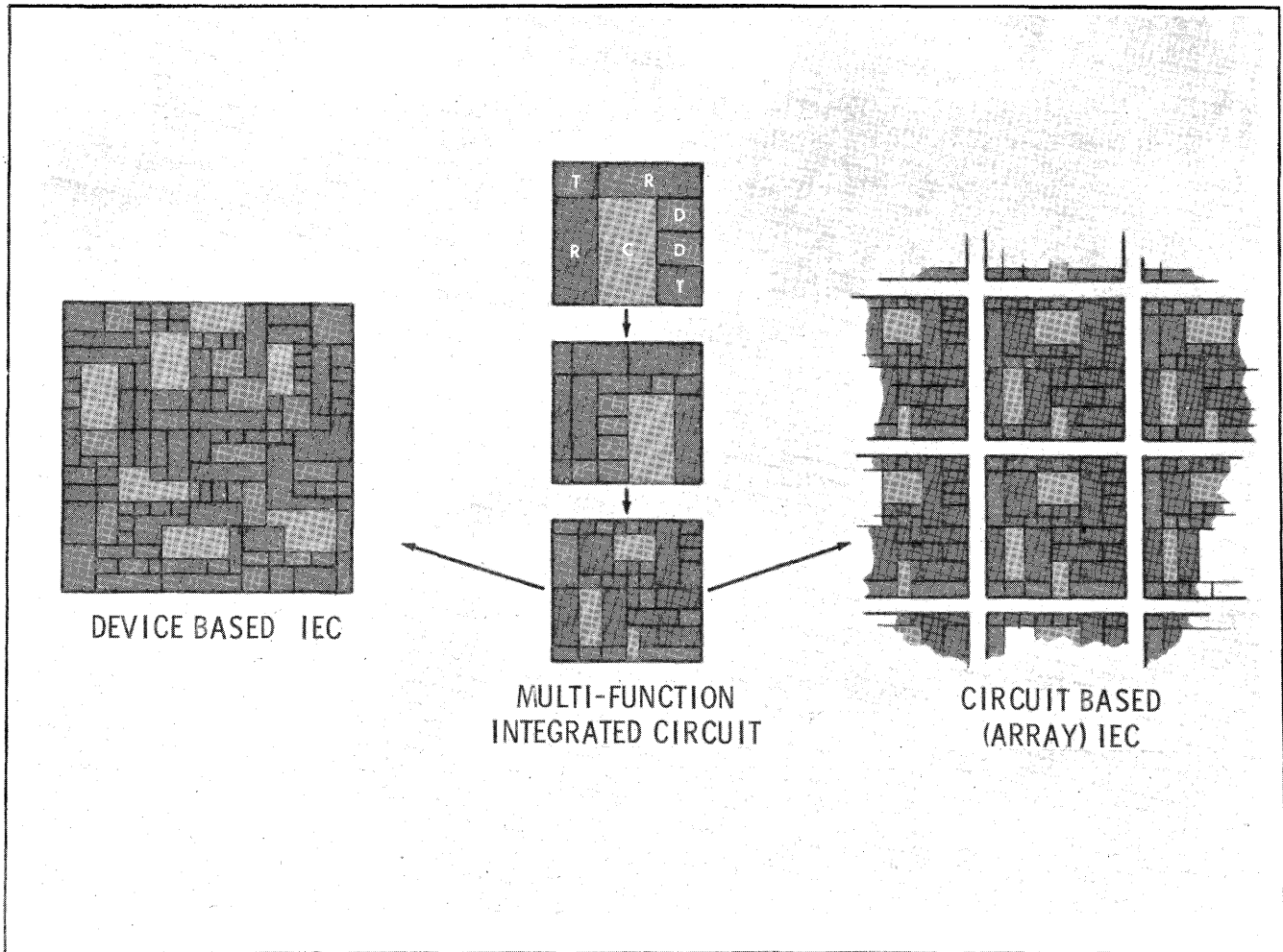


Figure 5. Evolution of integrated electronics.

cally on the right side of Fig. 5. Here, unit cells consisting of circuit functions such as NAND gates or flip-flops are the basic building blocks. This type of IEC is formed by interconnecting an array of unit circuit cells. We will use the term array for this approach.

The unit cell may be a simple NAND-NOR gate, occupying, for example, an area 10 by 10 mils. More than a single unit cell may be used and they may be intermixed. The step-and-repeat optical process allows for repetition and intermixing of the unit cells over the entire slice of silicon.

The distinctions between the device-based and circuit-based (or array) approach to large-scale integrated electronics will be developed in more detail in a later section of this paper ("Discussion of Selected Aspects"). Before doing this, we will discuss basic devices for use in large-scale integrated electronics.

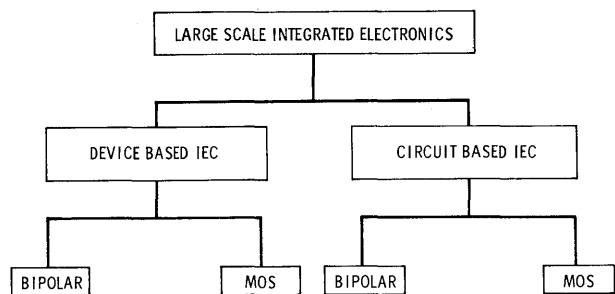


Figure 6. Large-scale integrated electronics.

BASIC DEVICES FOR USE IN LARGE-SCALE INTEGRATED ELECTRONICS

Two active device structures that will be considered for LSI application are the bipolar transistor and the MOS transistor.

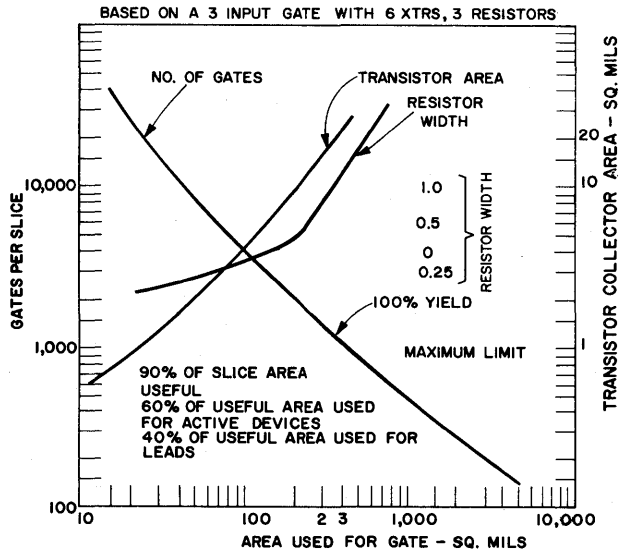


Figure 7. Bipolar transistor and gate densities for 1-inch slice diameter.

Bipolar Transistor

The bipolar transistor has made the transition from a two-sided device to a one-sided device suitable for integrated circuits in an effective way.

Numerous techniques have evolved, from triple diffusion to dielectric isolation, for processing one-sided transistors of high performance. The area requirements for transistors have decreased consistently over the past five years. Fig. 7. shows that 10,000 to 50,000 gates can be fabricated in a 1-inch slice of silicon by providing transistors of smaller collector area along with smaller resistors. Thus, bipolar transistors can provide for the high density packing required for large arrays.

MOS Transistor

The MOS transistor is outlined in Fig. 8 and some of its key properties are summarized in Fig. 9. It is inherently a single-sided device, self-isolating, and occupies a small area. The MOS device can be connected to form a load resistor as shown in Fig. 10, and in Fig. 11 the area is plotted that is required to achieve values of resistance by MOS, diffused and thin film technology. Figure 11 shows that the MOS device is a convenient way to achieve impedance levels of 100 kΩ in a small area. The ability to fabricate low-power, medium-speed circuits in a small area using MOS active devices and MOS load

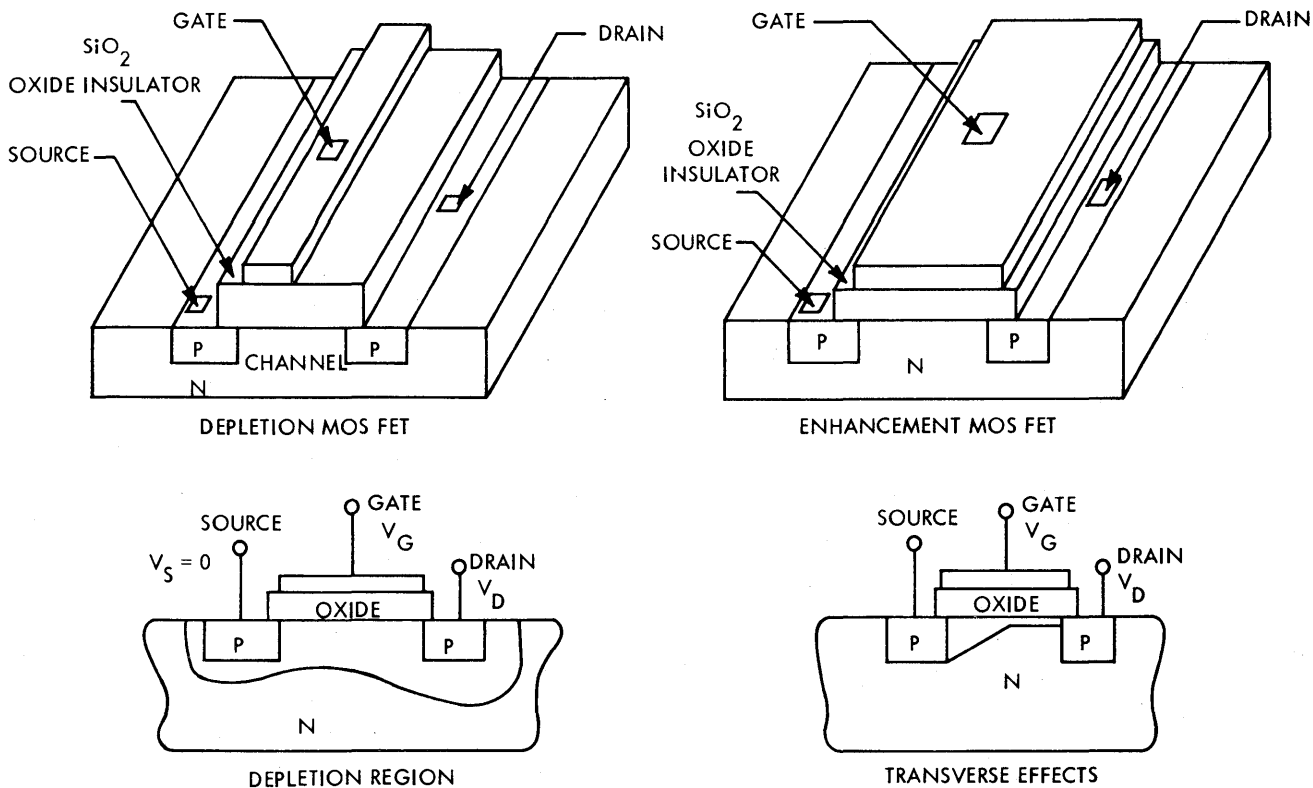


Figure 8. MOS enhancement and depletion models.

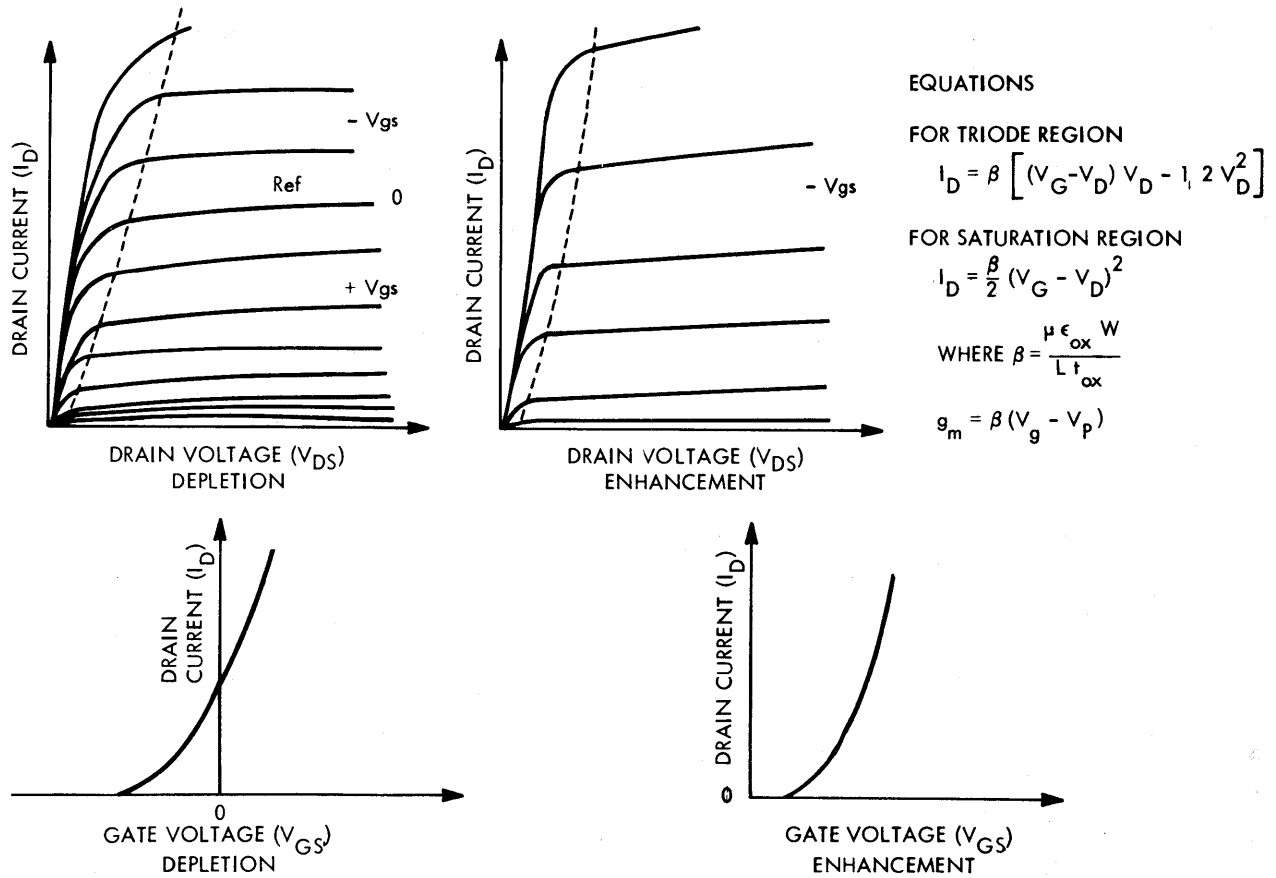


Figure 9. Characteristic curves for MOS transistor.

resistors is an attractive application of MOS technology.

Another promising application of MOS technology is in the use of N-channel and P-channel MOS devices in complementary circuits as shown in Fig. 12. This configuration will provide switching speeds in the 25–50 nsec region, with extremely low DC power drain (0.01 μ W). However, it does not have the processing simplicity of the single-polarity MOS structures.

Applicability of Bipolar and MOS Transistor for LSIE

With these general characteristics of both MOS and bipolar at hand, let us now attempt to assess the merits of each for specific LSI applications. A first consideration is the device densities that can be achieved, leaving aside for the moment the question of yield. Figures 13, 14 and 15 show the areas required in terms of a fundamental width W for the bipolar device with load resistor, the MOS device (assuming it also is used as the load), and the basic inverters using the two devices. From these considerations device densities are forecast as shown in Fig. 16. This figure shows that single-polarity MOS is capable of higher device densities than bipolar by at least a factor of two. A principal reason for this is that no isolation diffusion is needed for MOS; it is inherently self-isolating. The figure also shows that today we are working with MOS device densities of 100,000/in² and bipolars of 50,000/in.² The figure also forecasts a density limit of about 10⁶ devices/in² because of interconnection area requirements.

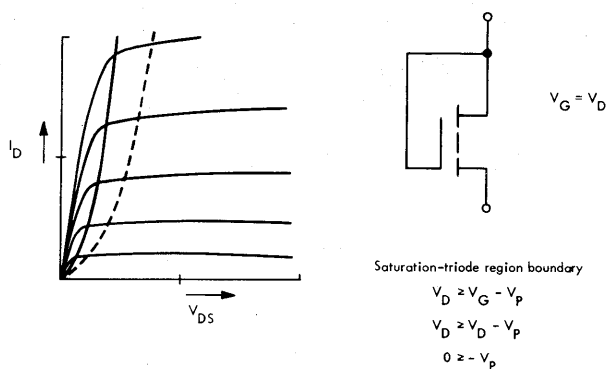


Figure 10. MOS connected as a load resistor.

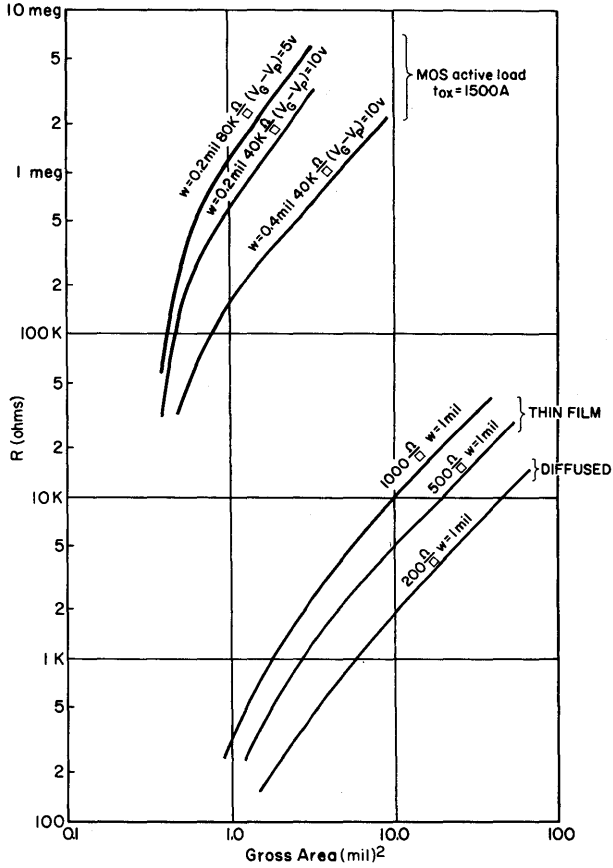


Figure 11. Load resistance vs slice area for MOS active load, thin film, and diffused resistors.

The lower density of the discretionary curve is because of the area used by redundant devices.

A second consideration between MOS and bipolar is the speed-power relationship per unit of silicon surface area. Analysis⁷ results in Fig. 17, which shows a clear superiority in bipolar where speed

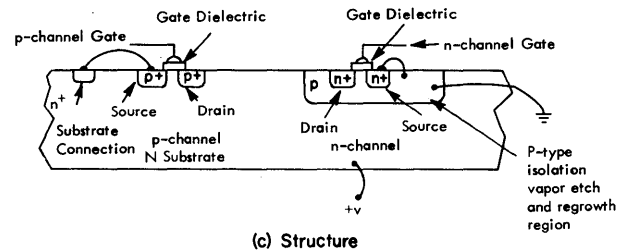
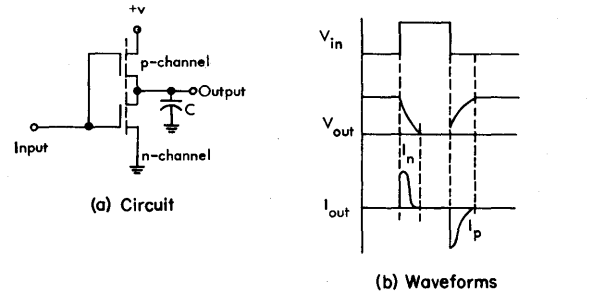


Figure 12. Complementary MOS structures and circuits.

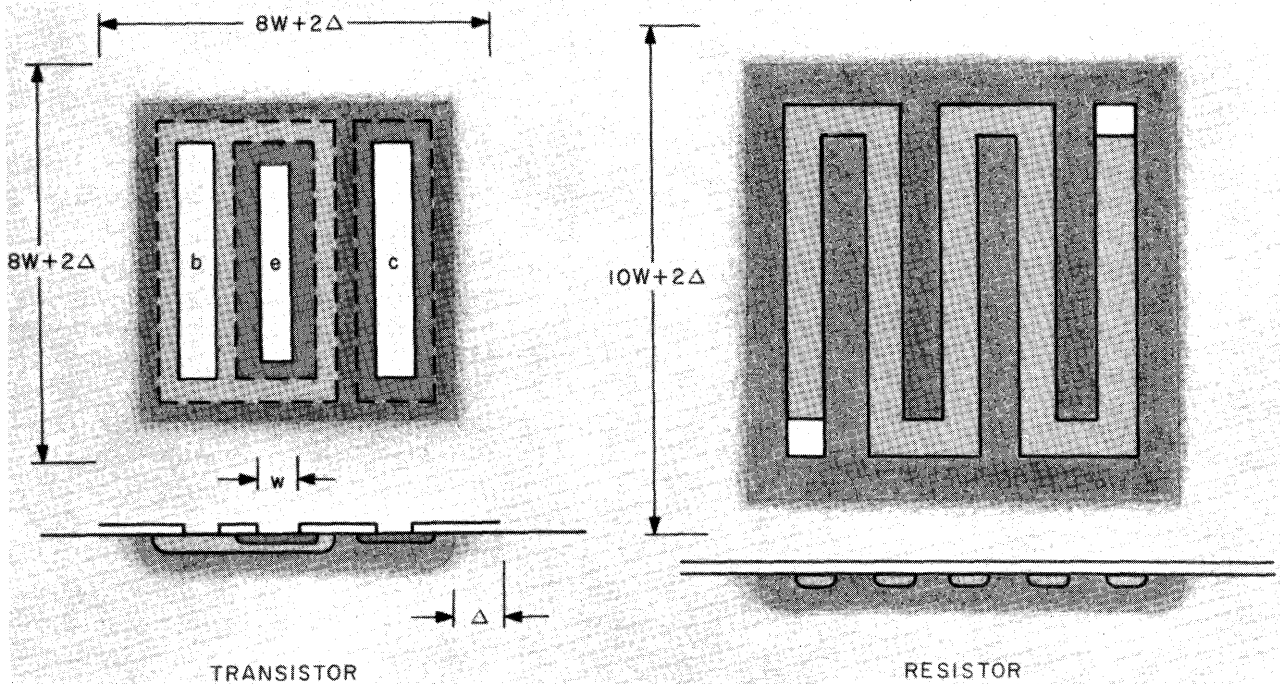


Figure 13. Area for bipolar transistor and load resistor as function of resolution width W .

and/or power is required. This superiority relates to the inherently higher gain (transconductance) of bipolar over MOS. The importance of g_m is that it is a measure of a device's ability to charge capacities, which in turn is a measure of a device's switching speed. Figure 17 shows that at $1\mu\text{A}$, for the same area A and capacity C , the g_m of a bipolar transistor is $40\ \mu\text{mhos}$, while that of an MOS transistor is $4.5\ \mu\text{mhos}$. To achieve a g_m of 40 for the MOS requires an area and capacity increase of 100 times. At higher currents the superiority of the bipolar over the MOS transistor is even greater.

From the two comparisons developed above, device densities and speed-power relations, we can reach some general conclusions as to the applicability of MOS and bipolar technologies. For those applications where MOS has sufficient speed and current handling capability, it should win out on the basis of achieving higher complexity per unit of chip area. Examples today include shift registers in the

megacycle speed range where the capacity loading of devices is small because the fan-out is basically one. Some remarkable achievements have already been made in employing serial logic using MOS for small processors such as desk calculators.

On the other hand, bipolar will be the choice for parallel logic organization, particularly if speed is a factor. The basic advantage of the bipolar is its inherently high transconductance (g_m); therefore, it is superior where appreciable capacitance must be charged as in parallel logic.

We forecast a large applications area for both technologies, and as our technological capability increases to where both kinds of devices are processed together on the same slice in monolithic structures, another very large area of application. Finally, while our discussion has been limited to bipolar and MOS device structures, many other device types, e.g., Schottky-barriers, will be integrated into LSI structures.

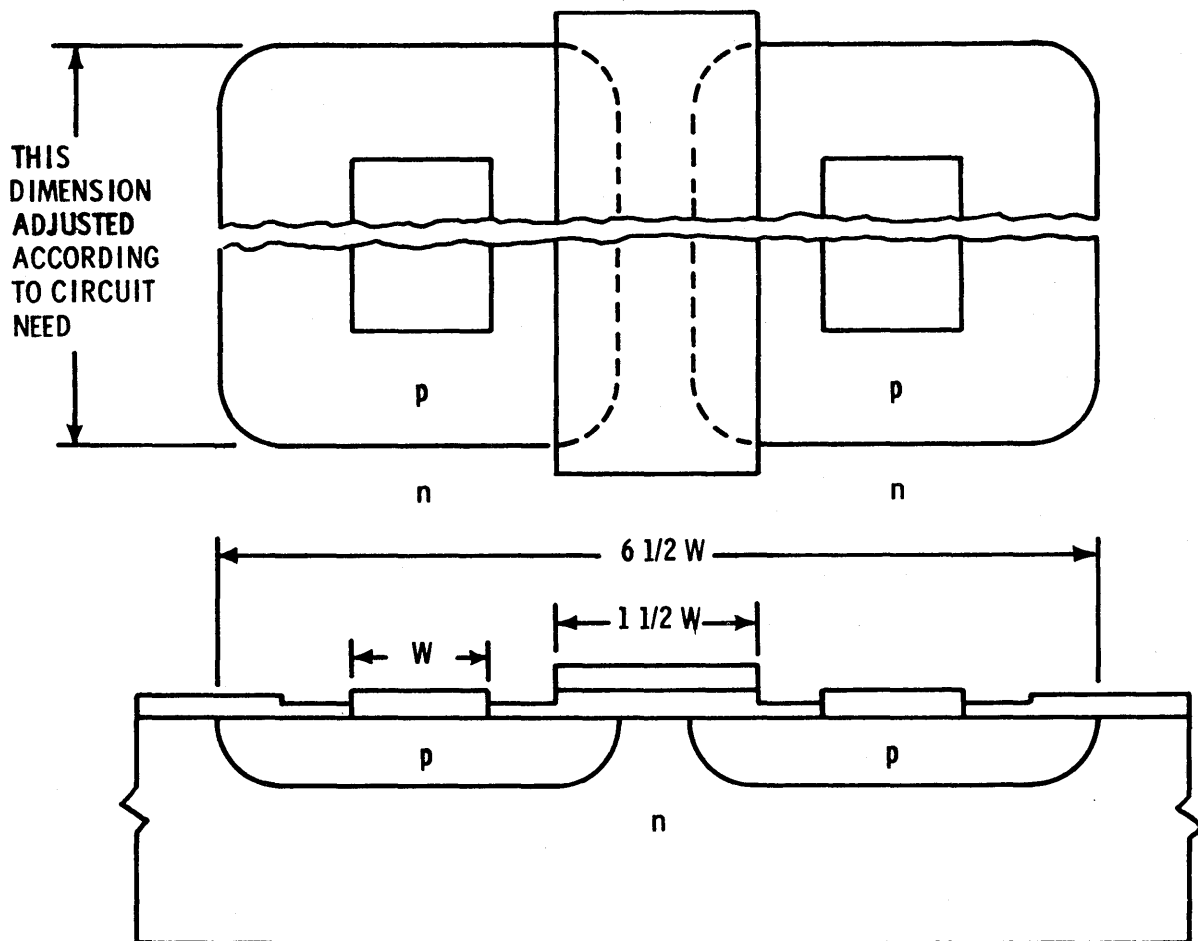


Figure 14. Area for MOS transistor and MOS resistor as function of resolution width W .

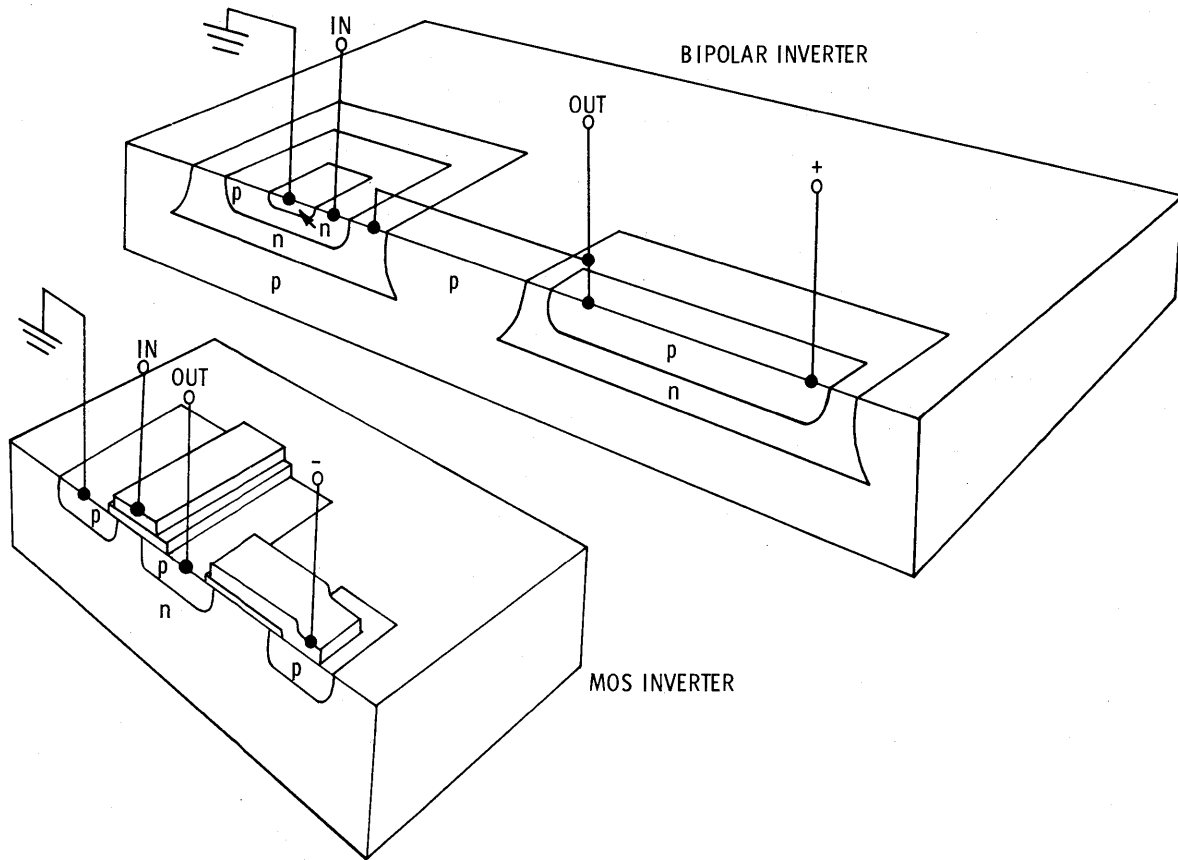


Figure 15. Basic inverters for MOS and bipolar.

DISCUSSION OF SELECTED ASPECTS OF LARGE-SCALE INTEGRATED ELECTRONICS

We now discuss key aspects of LSI, including more detail on the device-based and circuit-based approaches, discretionary wiring, forecast of level of integration to be achieved over the next 10 years, design by computer, standard products versus flexibility to custom requirements, and special requirements for subnanosecond arrays.

Device-Based Approach to LSI

The device-based approach is the natural extension of multifunction integrated circuit design and differs from the latter in that equipment components are designed and processed rather than circuit functions. Consider for example the design and processing of a 50-bit MOS shift register. The design is worked out using MOS active devices and MOS load resistors in an optimum fashion, without any attempt to partition the shift register into circuit building blocks.

The chief advantage to this approach is that one can achieve a very high density of devices, or conversely, achieve an IEC in a small area. Table 1 and Figures 18a and 18b show typical IEC's made from bipolar and MOS devices: MOS achieves higher device density, but at lower speed as discussed above.

The entire IEC, or at least a major part of it, is achieved within a relatively small chip size. A characteristic of this approach is that 100% yield is required over the chip area (e.g., 100×100 mil), but not over the entire slice. Consider for example a 1.50-inch diameter slice where

$$\begin{aligned} \text{Number of chips} &= \frac{\text{slice area}}{\text{area/chip}} \\ &= \frac{\pi 9/16}{100 \times 100 \times 10^{-6}} = 175 \end{aligned}$$

If 50 of these 175 chips are good, the overall material yield is 35%.

While discretionary wiring techniques are not used within the chip areas (100% yield is demanded here) it is possible that discretionary wiring will be used

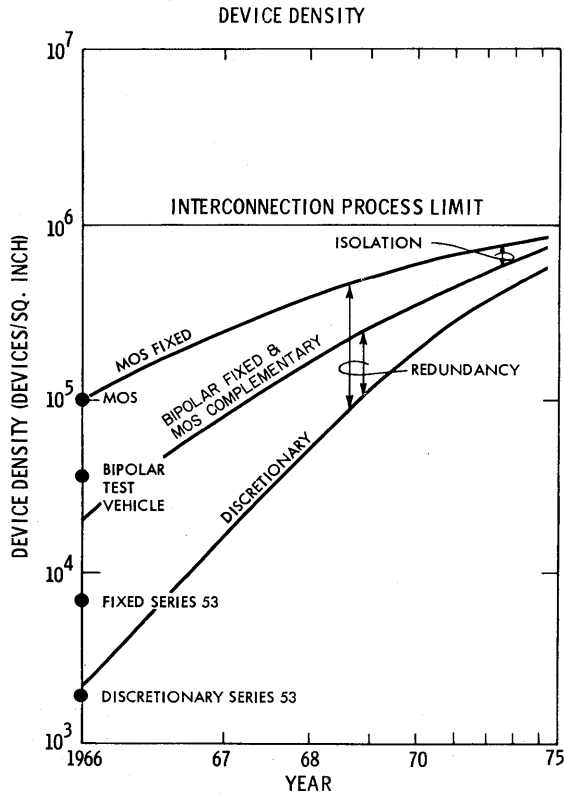


Figure 16. Ten-year forecast of device density.

to wire together N good chips in a slice to form a more powerful function. In the example above, the 50 good chips might be connected together without scribing the slice into individual chips. One reason for doing this is that the oxidized silicon surface provides an excellent surface on which metallic transmission lines can be deposited.

What are the chief handicaps or limitations of the device-based design approach to LSI? Probably the most important limitations are its lack of adaptation to change and long-time cycle for implementation. Each IEC design requires a complete set of masks, including diffusion and metallization. While some flexibility can be attained at the metallization level by providing extra devices in the chip (master slice concept), the approach is most suitable for standard products where relatively large production runs occur, or for custom designs of high volume.

In summary, the device-based approach to large-scale integrated electronics is already off to a fast start. For the case of bipolar technology, the IEC's are being designed to give added logical power and capability to existing IC product lines, as for example, three of the TI units listed in Table 1 augment the Series 54 line. For the case of MOS, the

leading edge of this technology has aimed towards serial logic systems designed around shift registers.

Array or Circuit-Based Approach to LSI

In this approach circuits are used as the basic building blocks for designing and processing the IEC's. A number of advantages occur when one moves the building block level from the device to the circuit function. A key advantage is that Boolean logic equations are readily expressible in terms of NAND, NOR and related logical decision circuits. These equations, which are basic to the design of computers, are independent of the devices underlying the Boolean circuit element.

Another advantage is that it is possible to incorporate a high degree of flexibility into the process technology. Consider the problem of an IEC manufacturer responding to a customer's request for a specific IEC. Assume that the customer presents the IEC manufacturer with Boolean equations and specifications. Let us examine the different methods (see Table 2) by which the manufacturer may respond. We will assume that silicon slices are

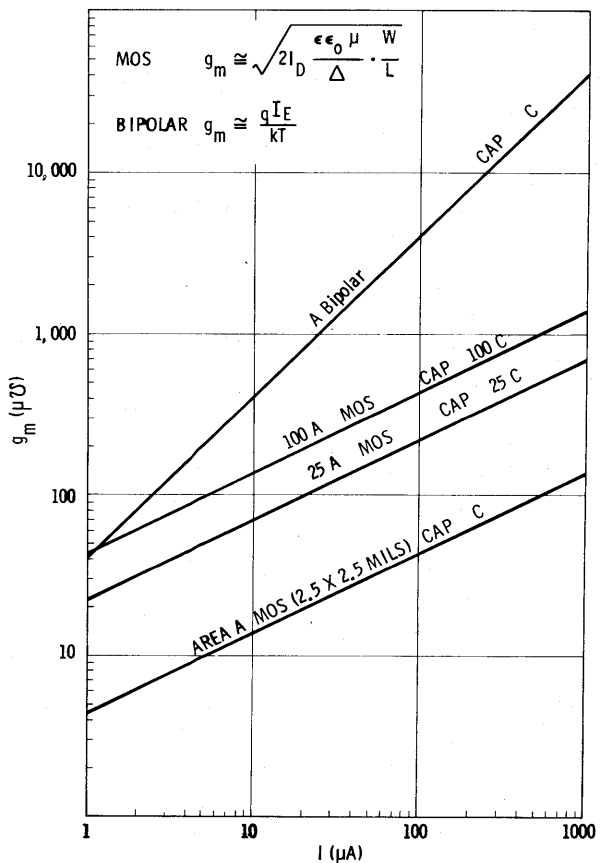


Figure 17. Transconductance g_m of bipolar and MOS transistors vs current and device area.

Table 1. Integrated Equipment Components—1966

Mfr.	Component	Device	Area <i>in</i> ²	No. of Devices	Device Dens. <i>devices/in</i> ²	Speed <i>nsec</i>	Power <i>mW</i>	Pads
TI	Series 53 Array	Bipolar	0.7	1,200	1,720	35	1200	60
TI	8-Bit Shift Reg	Bipolar	0.006	160	26,500	15 mHz	190	6
TI	Honeywell Memory	Bipolar	0.0071	100	14,000	25	250	14
TI	Parallel Load Serial Shift	Bipolar	0.01	150	15,000	25	270	22
GME	A-C 100-Bit Shift Reg	MOS	0.0065	613	94,500	1 mHz	200	12
GI	D-C 21-Bit Shift Reg	MOS	0.0042	158	37,600	500 kHz	150	11
TI	22-Bit Shift Reg	Bipolar	0.0126	350	27,800	3 mHz	35	8
TI	B to D Decoder	MOS	0.0057	152	26,700	200 kHz	25	26

processed with 100% yield of unit cells—the question of dealing with cells on the slice that are not good will be discussed below, under “Discretionary Wiring.”

For those requirements where the logical function (IEC) can be built by connecting together identical unit cells (e.g. NAND gates), the manufacturer need only make masks which provide for interconnecting gates in a specific pattern. Processed slices would be on hand which have a large number of NAND gates. Such slices would have a first level of metallization on them as indicated in Category I, Table 2. Only new masks for 2nd and 3rd level metallization are required to meet a customer's specific request. More flexibility can be incorporated into this approach by providing more than one kind of unit cell on the slice. For example, a slice with a mixture of gates and flip-flops will meet a large proportion of computer requirements. This approach provides very fast response to customer's needs.

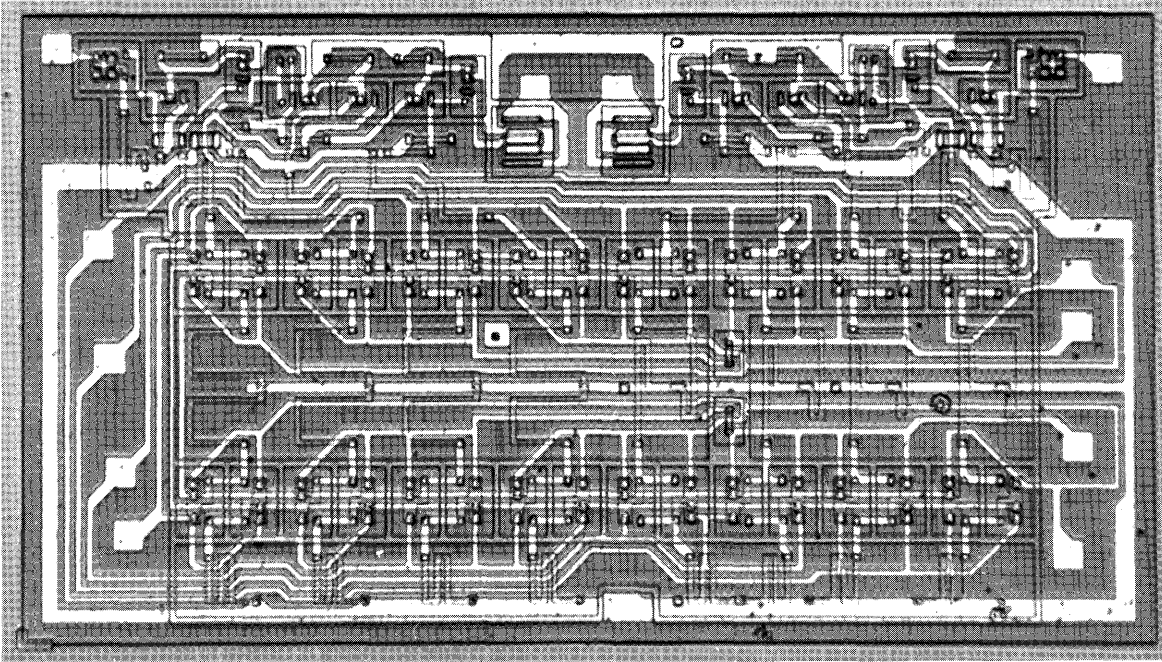
A second approach, Category II, involves processing slices which have a common master unit cell through the diffusion and third oxide removal operations. This master unit cell would have sufficient devices such that 10 to 15 different logical circuits could be attained by variations of the first level of metallization. Coupling this with the ability to interconnect the unit cells together in specific ways provides a high degree of flexibility. However, this requires a first-level metallization mask as well as a second (third if necessary) level mask.

The third approach is to generate a complete set of masks for each order, as in Category III, Table 2. A complete set of diffusion masks are required, along with first level metallization masks (second and third if necessary). This is the most expensive approach

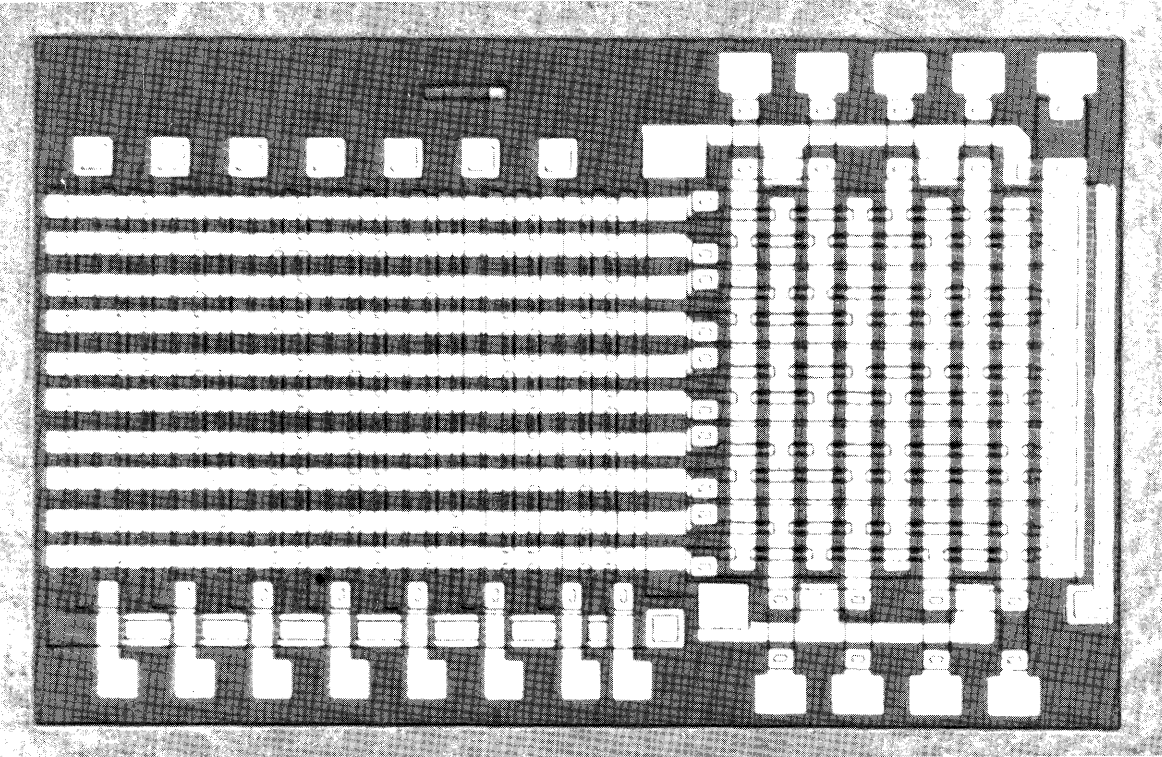
Table 2. Categories of Array Approach to LSI

Process	Definition Category		
	I	II	III
Oxidation	1st Oxide Removal (OR)	1st OR	1st OR
Collector Diffusion	2nd OR	2nd OR	2nd OR
Base Diffusion	3rd OR	3rd OR	3rd OR
Emitter Diffusion	4th OR	4th OR	4th OR
Metal Deposition	1st Level Leads	1st Level Leads	1st Level Leads
Insulation Deposition	2nd Level Insulation	2nd Level Insulation	
Metal Deposition	2nd Level Leads	2nd Level Leads	
Insulation Deposition	3rd Level Insulation		
Metal Deposition	3rd Level Leads		

Operations in boxes permit component variability.



a



b

Figure 18. Examples of device-based IEC's—characteristics in Table 1. (a) Eight-bit bipolar IEC. (b) Binary-decimal decoder MOS IEC.

from the initial design phase and the most time consuming, but it can provide savings in materials usage and therefore may be favored for longer production runs.

A comparison of the approaches of Table 2 reveals that Category I provides the fastest turn-around time and lowest initial cost but has the poorest utilization of unit cells in the slice. Category III provides the best usage of area in the slice because it has no cells or devices that are not used, but is the slowest in turn-around time and has the highest initial expense. Category II is midway between I and III in all respects.

All three categories are under development in the semiconductor industry. Prediction of the relative costs and degree of usefulness of these approaches is difficult at this time because of the embryonic state of the technology.

Discretionary Wiring

The goal of processing technology is 100% yield of unit cells over the slice. For the same reason that integrated circuit yields are much higher than were expected, considering arguments based on discrete device extrapolations, array yields will be much higher than predicted from simple extrapolations of integrated circuit yields. However, at some array size one can expect that defective unit cells will become a problem which will affect overall array yield. It is desirable then to develop a system whereby defective cells can be omitted from the interconnected array.

The problem of achieving discretionary wiring at low cost relates to testing, automatic mask making, and computer programming of interconnection systems. It should be emphasized that successful implementation of the discretionary approach demands a rapid, low-cost mask-making procedure since each slice may require a different interconnection pattern because the "good" cells will occur in different locations. Figure 19 shows an approach to discretionary wiring which is being developed at Texas Instruments Incorporated under sponsorship of the Air Force Systems Command, Wright-Patterson Air Force Base.⁸ At the top left of the figure a silicon slice is shown which has an array of unit cells. The unit cells consist either entirely of NAND gates and flip-flops, or a mixture of the two. A first level of metallization is provided on these cells so that they can be probed by multiprobe test equipment. The

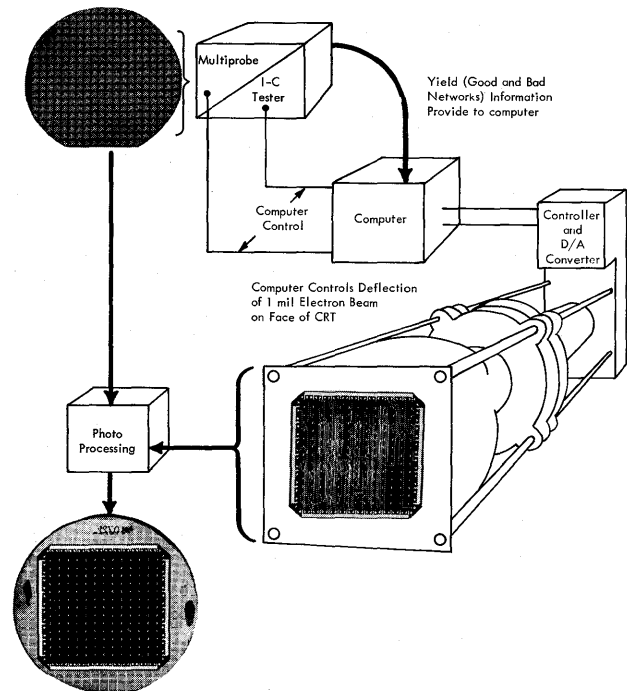
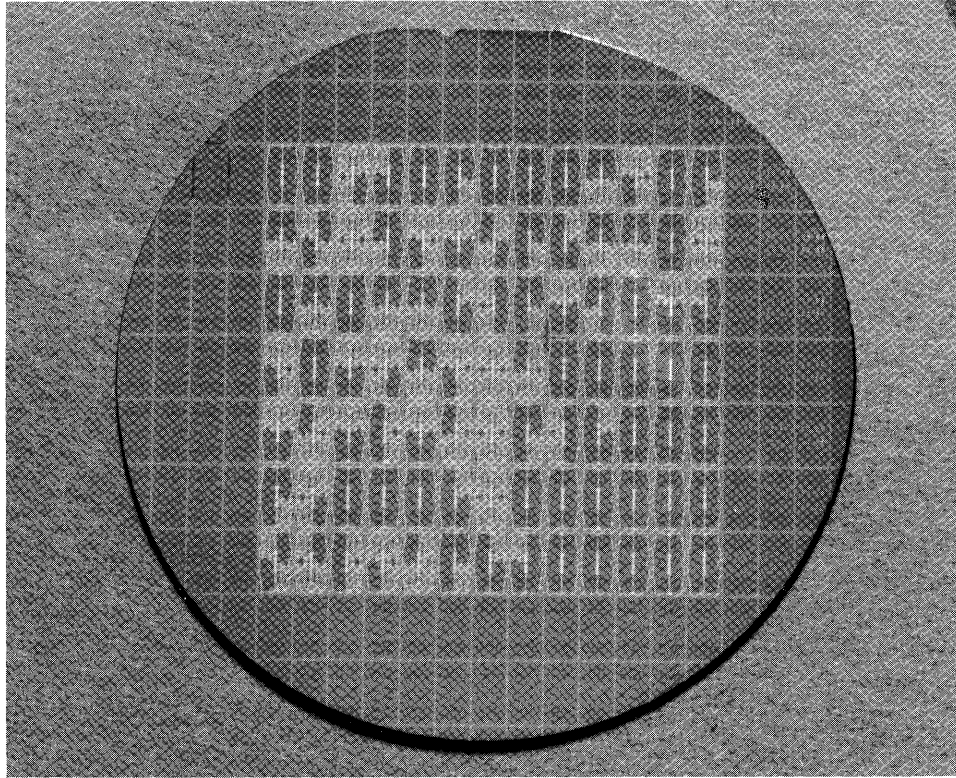


Figure 19. Discretionary wiring system being developed by Texas Instruments under Air Force sponsorship.

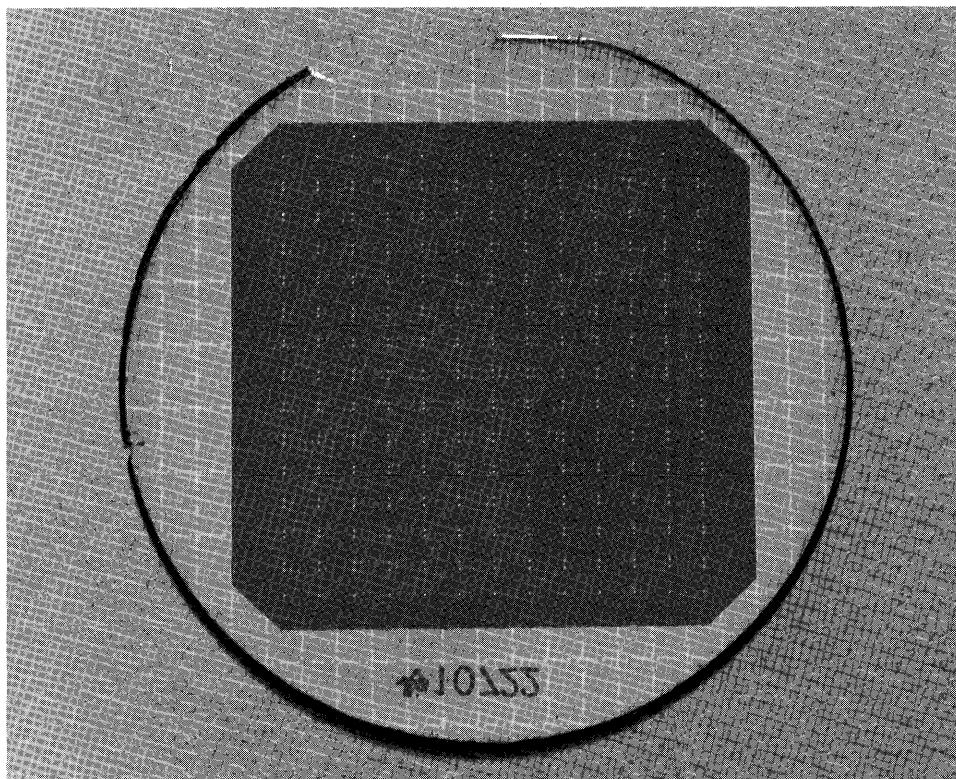
information concerning location of good and bad cells is fed into the computer, which generates an interconnection pattern. The control for each pattern is fed to the high-resolution CRT, and a pattern is generated on the face of the CRT. This pattern is projected on photosensitive material to form a photo-mask. Finally the set of masks is used to process interconnection patterns on the semiconductor slice.

All elements of this system are under active development. It is planned to have the system in operation during the last half of 1966.

To illustrate the discretionary wiring approach the series of pictures in Fig. 20 are given. Figure 20a shows a slice where the "good" gates to be used in the array have the first level of metallization applied. Each rectangular area contains four Series 53 gates, and the full array consists of 120 "good" gates. The slice with a layer of insulation applied and holes etched through to the first level metal are shown in Fig. 20b. Horizontal interconnections which were designed by the computer are shown in Fig. 20c. Figure 20d shows another level of insulation applied and holes etched through to the second level of metal. Figure 20e shows the third and final level of metallization which completes the interconnection of the 120 gates.

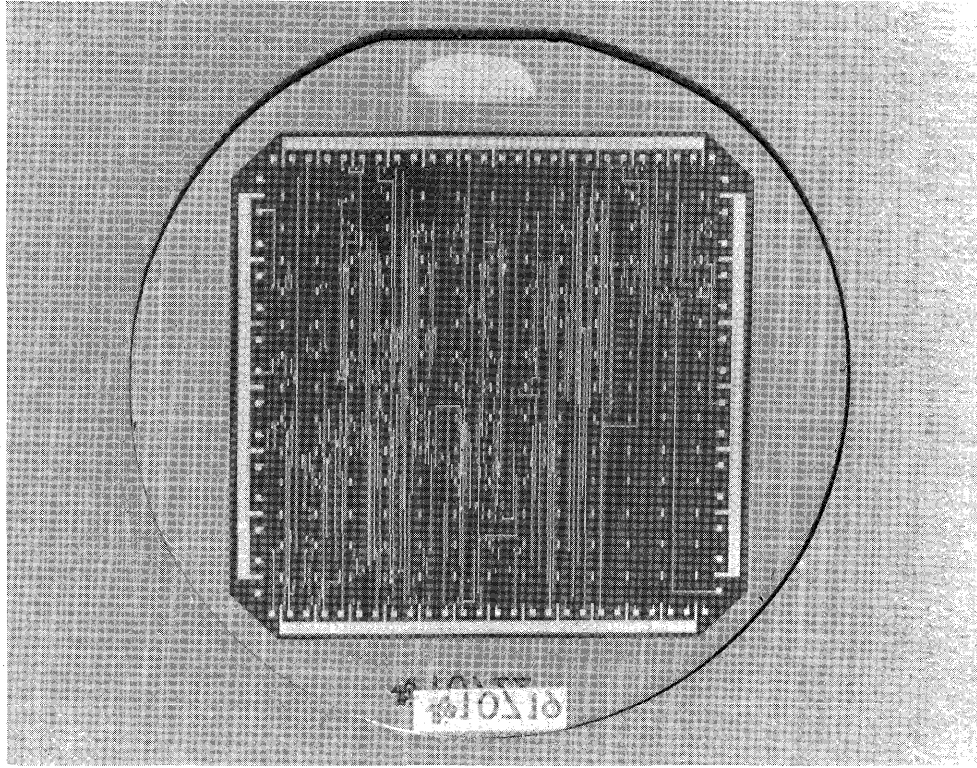


a

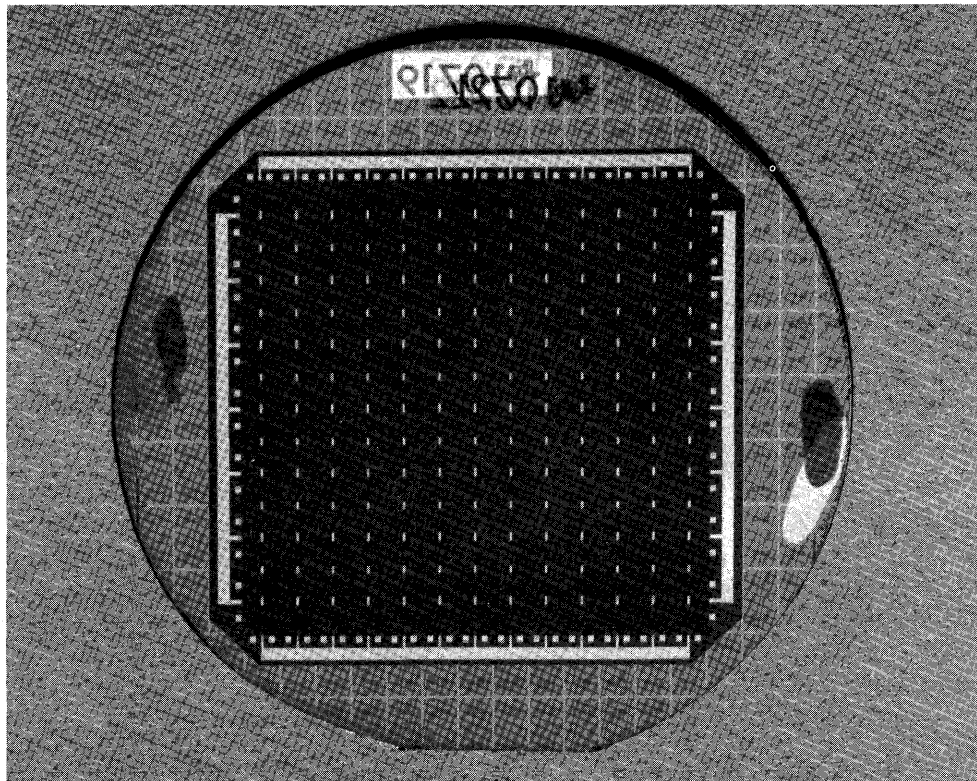


b

Figure 20. Series 53 array of 120 gates (process description in text).

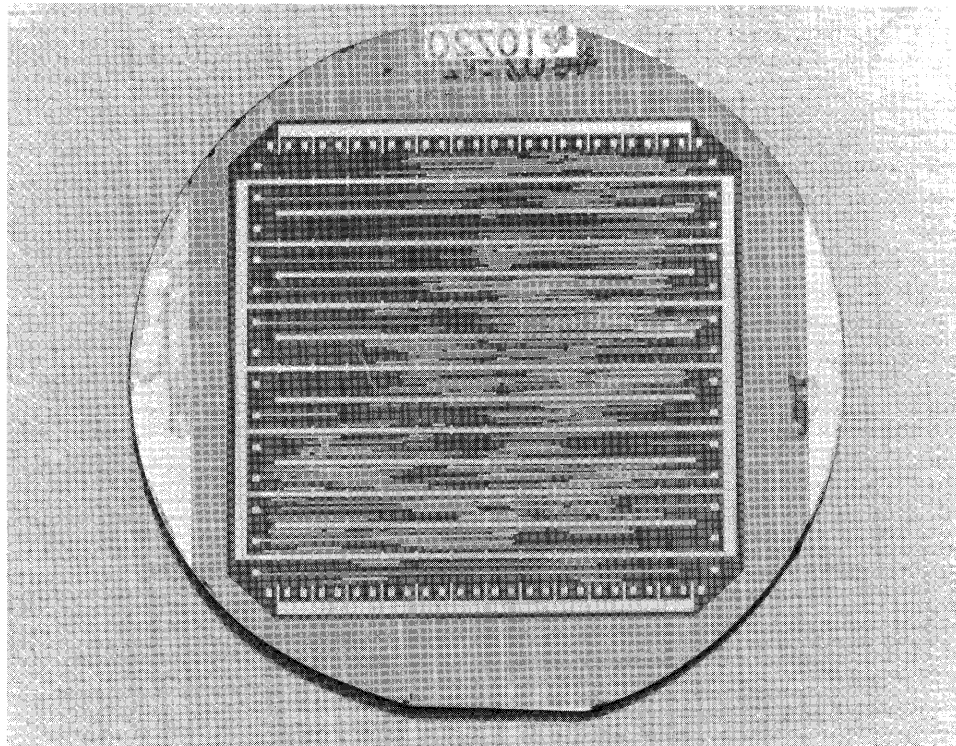


c



d

Figure 20. *continued*



e

Figure 20. *continued*

While the masks for the above example were cut by normal techniques, the system shown in Fig. 19 will make this an on-line process of short-time response. Characteristics of this array (the Series 53 array) are shown in Table 1. Comparison shows that this array has nearly 10 times the number of bipolar devices than the other bipolar IEC's, which are of the 100% yield category.

An important aspect of the discretionary approach is that it will allow for integrating to higher levels of complexity at any given time than will the 100% yield approach. For example, the aforementioned program has a minimum goal of 250 gates per slice or 2500 devices per slice, and a maximum goal of 1000 gates per slice or 10,000 devices per slice; furthermore, the arrays will be ready for production by the end of 1967. It is believed that this is an order of magnitude higher performance than will be achieved by 100% yield methods in the same time period.

We note that this approach to discretionary wiring has much in common with the process of Category I in Table 2. The interconnection pattern generator can be used to generate the second and third level masks required in Category I. If one has 100% yield over

the area of the slice to be used, the probing step is eliminated. Also, the computer programs for pattern generation will be somewhat simpler because the location of gates is known.

Forecast of Degree of Integration of IEC's

Our forecast of device density in Fig. 16 over the next 10 years was relatively simple to make since it was derived from reasonably well-defined parameters. In contrast, the forecasting of the level of integration of IEC's over the next 10 years is a much more complicated and less precise task. Subjective as well as objective points must be considered. However, it seems worthwhile to discuss some of the aspects of the problem and to arrive at a "forecast," even though admittedly it is unprecise.

Let us first define what we mean by level of integration. By this we mean the total number of good devices that will be interconnected on a single monolithic chip of semiconductor material (presumably silicon). We will not discuss the "chipping within a package" approach.

A first consideration is the theoretical device density, shown in Fig. 16. The MOS and Bipolar

Fixed curves imply 100% yield or fixed interconnection pattern (FIP). The Discretionary curve indicates a lower density because of the area required for redundant devices.

A next consideration is the chip size itself. Here we must consider a number of factors including single field of view optical limitations, step-and-repeat optical techniques, crystal size, and process yield limitations.

Optical problems limit the area over which high resolution can be achieved in a single field of view. Figure 21 shows the capability of the best of today's lenses, and the 10-year forecast. A 100 × 100 mil area with 0.2 mil lines is representative of today's integrated circuits production. The area will increase to 300 × 300 mil and resolution should approach the wavelength of light. New techniques such as electron beam will be required to achieve further improvements, but we will not consider these possibilities in the discussion.

Step-and-repeat techniques allow for the stepping of a fixed field of view over much larger areas. This process is used in today's production technology such that 100 × 100 mil or smaller patterns are stepped over slices 1½ inches in diameter.

From these considerations Fig. 22 has been constructed. The area labeled Fixed Field Pattern forecasts that fixed pattern IEC's will increase from the present 100 × 100 mil size to 300 × 300 mil size.

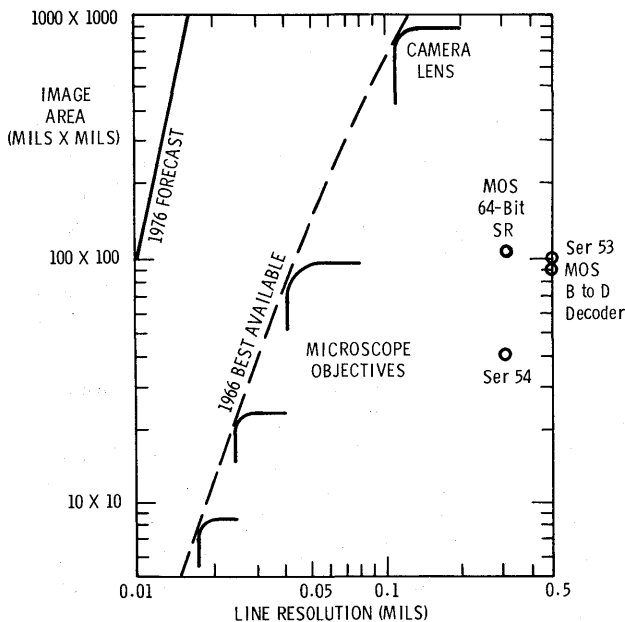


Figure 21. Optical image dimension as a function of resolution.

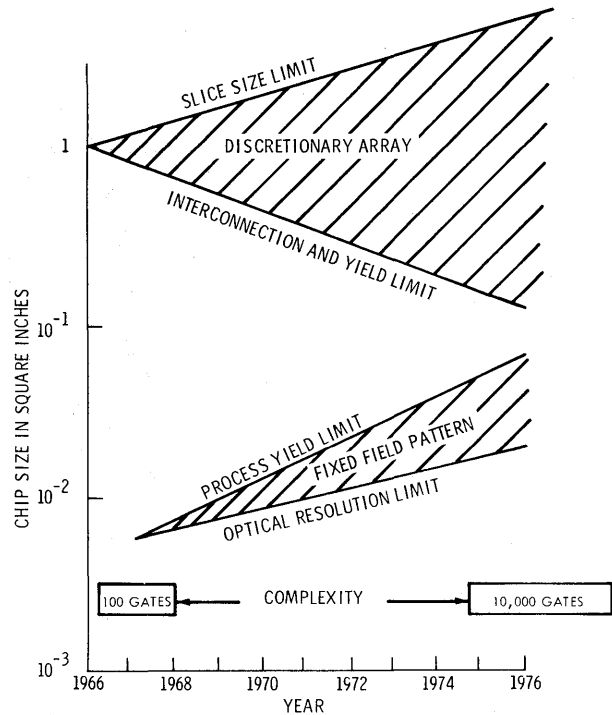


Figure 22. Chip size forecast.

The smoothness of the curves relates to the gradual increase in area over which 100% yield is achieved.

The upper shaded area is based on step-and-repeat optical techniques and discretionary wiring. The upper line shows the increase in crystal size from 1-inch diameter to 3-inch diameter material. The lower line suggests that interconnection and yield improvement will allow for smaller chips to be used because of less area required for redundancy.

Finally Fig. 23 shows the author's best judgement as to the number of devices per chip that will actually be used over the next 10 years. These curves have been developed by consideration of the factors of Figs. 16, 21, 22, and 23, by knowledge of today's capabilities (1966 data points of Fig. 23), and by the subjective consideration that 100,000 devices per chip, which will provide logic power of 10 K to 20 K gates, is a practical requirement limit. One argues that 10 K to 20 K gates provide suitable basic building blocks for computer systems.

Note that the distinction between discretionary and fixed patterns disappears in the 10-year period. It is reasonable to forecast that 100% yield will be achieved for this complexity in this period. This conclusion does not invalidate the present program on discretionary wiring. As Fig. 23 shows, discretionary wiring technology will provide for the more rapid

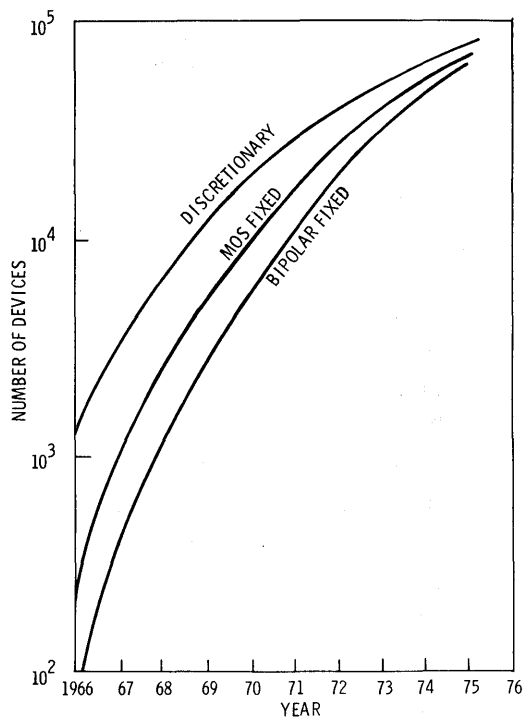


Figure 23. Forecast of number of devices per chip—IEC complexity.

development of the high-complexity IEC's that are vital for the industry.

The merging of bipolar and MOS techniques does not invalidate our previous discussion—the 100 K MOS devices will be achieved in a smaller area than that required for the bipolar devices (Fig. 22).

Our rather arbitrary limit of 100 K devices per chip does not imply a technological limit so much as a practical limit. The direction of digital system requirements, which we shall not attempt to forecast, could generate motivation to increase this limit markedly.

Design by Computer

At the levels of complexity that are forecasted for IEC's, the problem of design is a formidable one. For this and other reasons we can expect that computer-aided design will be developed. One approach is to use computer-aided design directly at the device level; that is, design IEC's by computer directly from device parameters. This is what the human does today when approaching LSI through device-based design.

However, it is likely that a more useful approach will be to use computer design at the circuit func-

tion level. Here one defines a set of logical circuits such as NAND gates, shift registers, flip-flops, etc., and utilizes a computer to design layouts which minimize crossovers, area, etc.

An important benefit of design by computer in terms of circuit building blocks is that it will shorten the reaction time of IEC manufacturers to system requirements. The systems manufacturer will state his requirements to the IEC manufacturer in terms of Boolean logic equations. The IEC manufacturer will translate these requirements to process steps in the factory using computer-aided design.

Another reason for emphasizing design by computer is that in some cases, normal breadboarding techniques will not simulate sufficiently well the actual conditions on the silicon slice. For example, MOS devices, if breadboarded as discrete devices, would be so heavily loaded down with capacity that their performance would bear little relation to that in a monolithic structure. Subnanosecond bipolar transistors, if breadboarded as discrete devices, would have such large delays between devices as compared to the monolithic case, that information gained by breadboarding would have little value.

Finally, we note that back panel "patching" techniques are not available in highly integrated structures, and so the elimination of human errors becomes especially important.

Standard Products vs Flexibility to Customer's Requirements

A key question of large-scale integrated electronics is—To what degree will standard product lines of IEC's be accepted by the equipment and systems manufacturers, or conversely, to what degree will they demand custom IEC's? Before attempting to answer this, it is perhaps worth examining what has happened in integrated circuits.

In the early days of integrated circuits considerable resistance was given to accepting the idea of standard circuits. Important custom designs have been and are continuing to be developed, and often these become the technological base for standard lines introduced at a later date. However, it is clear today that there has been far greater acceptance of standard lines by the industry as a whole than was originally predicted.

Today one hears similar arguments. For example, because integration to the equipment component level necessarily involves the computer logic, some pro-

ponents claim that IEC's will be entirely a custom business. It is the author's opinion that this will not be the case. There certainly will be considerable custom design work, particularly in the early stages of the development of the technology. However, as time goes on, the author forecasts that standard IEC's will be developed and produced that will be basic building blocks of systems. Furthermore, it is the author's forecast that we will see an even greater acceptance of standard products at the IEC level of integration than at the IC level of integration.

Special Considerations for Ultra-High Speed (Subnanosecond) Arrays

Designers of high-speed computers are faced with two fundamental problems in making computers that switch appreciably faster than about 5 nsec. The first problem is that of the transmission time between discrete components. Remembering that electromagnetic signals travel 6 inches in one nsec, one recognizes that packing densities achievable by discrete devices run into phasing problems at about 5 nsec. Multifunction circuits provide some gain here because of the increased packing density. However, the improvement gained is marginal because each gate must be capable of communication with any other gate, at least on a single multilayered board.

The second computer design problem is also very fundamental: terminated transmission line structures of low impedance ($\cong 50 \Omega$) are needed in order to interconnect gates operating in the few nsec range. This configuration is required to prevent false reflections. Devices of substantial current handling capability are thus required which, in turn, limits the density to which circuits can be packaged together.

Array technology provides solutions to both problems. By interconnecting on the slice, several hundred gates can be located within one inch of each other. Thus, phasing problems will not be encountered on the slice, at least for speeds in the range down to 0.1 nsec. For devices 10 mils apart, the transmission time will only be of the order of picoseconds.

The second point is that the proximity of gates to one another (tens of mils) makes it unnecessary to provide low transmission line impedances between gates. Instead, the interconnections on the slice can be viewed as simple capacitors, ranging in the 1/10 pF and less range. Thus, current drive capability for

the active devices is reduced. This in turn allows for the appreciably higher packing densities required to minimize transmission delays.

This subject is further developed in Ref. 4. The conclusion is that LSI technology will make it possible to build computers which operate at decision switching speeds well below one nsec.

IMPACT OF LARGE-SCALE INTEGRATED ELECTRONICS

This technology promises major impact in many areas of electronics. A few of these are:

1. Lower cost data processing systems.
2. Higher reliability processing systems.
3. More powerful processing systems.
4. Incorporation of software into hardware, with subsequent simplification of software.

Pervasiveness of Electronics

A more general, very important result has been discussed by Patrick E. Haggerty in his keynote article⁹ in the Special Issue of the *Proceedings of the IEEE* on Integrated Electronics. Mr. Haggerty emphasizes that Integrated Electronics will result in electronics pervading our entire social structure. Quoting from Mr. Haggerty's article:

To say with Dr. Noble that electronics is a generic art, or for this author that electronics is *inherently pervasive*, is simply to say that the basic knowledge and the tools of electronics are so pertinent to the needs of our kind of society that the products and services which are the result of the knowledge and tools have nearly unlimited usefulness and can contribute in a major way across our entire social structure.

Yet, in spite of the pertinence of the knowledge and tools, there have been very fundamental limitations to our applying this knowledge and these tools as broadly as they justify and *realizing the inherent power and full pervasiveness of electronics*. Some of the most harassing have been:

- 1) The limitation of reliability
- 2) The limitation of cost
- 3) The limitation of complexity
- 4) The limitation imposed by the specialized character of and relative sophistication of the science, engineering and art of electronics.

The limitations are, of course, interrelated. Cost is obviously affected by the need for high reliability and necessarily complex solutions. Conversely, the

more complex the solution required, the greater the likelihood that reliability and/or cost will become a controlling limitation. Such solid-state devices as transistors and diodes have certainly led the way to marked improvement in reliability, but they have hardly eliminated complexity. The solutions we have achieved still have a relatively high enough cost to inhibit the application of electronics in those broad areas which we customarily describe as the industrial and consumer sectors of our economy. So far as the fourth limitation is concerned, electronics is indeed a sophisticated branch of engineering and as such it has required highly skilled practitioners. Yet the very sophistication called for inevitably limits the rate at which electronics can pervade our society. For electronics to be truly pervasive, it must be readily and commonly used by the mechanical engineer, the chemical engineer, the civil engineer, the physicist, the medical doctor, the dentist, the banker, the retail merchant, and by the average citizen in broader ways than just for bringing entertainment to his home. Electronics cannot be truly pervasive unless such persons whose needs call for the powerful tools of electronics are capable of using them. It hardly seems feasible to suggest that all these highly skilled practitioners in other professions must also become skilled in the internal complexities of ours. *The problem is considerably simplified, however, if the electronics skills which they require are limited to the comprehension and specification of the input and output parameters of the electronic functions they need.* And, it is exactly here that integrated electronics may prove to remove a large percentage of these communication limitations. The contributions integrated electronics is likely to make in removing limitations in the categories of reliability, cost, and complexity are also impressive. Indeed, because integrated electronics seems to have a high probability of removing an appreciable percentage of the limitations in all four categories, I believe it may bring the total of these limitations to a critical level. Subsequently, it may initiate the terminal phase in which electronics contributes in truly vital ways to all segments of our society.

Expanding upon the fourth limitation, large-scale integrated electronics, does indeed offer the promise of placing most of the problems associated with the specialized character of electronics in the hands of the materials technologist. The technologies described above should result in processed slices of semiconductor material; wherein the great majority of devices and internal connections are made by material processing. The terminals brought out of the packages will be functional in nature and relatively easy to work with. The inherently low cost and high reliability of Integrated Equipment Components should, along with the elimination of complexity and specialized "character of . . .," result in electronics pervading our entire social structure.

Table 3. Generations of Electronics

Generation	Basic Product	Limitations of Electronics
First	Tubes	Cost
Second	Transistors	Reliability
Third	Integrated Circuits	Complexity
Fourth	Integrated Components Equipment	Specialized Character of . . .

Fourth Generation of Electronics

Because of the dramatic nature of large scale integrated electronics it seems appropriate to define it as the fourth generation of electronics. Table 3 suggests that the first generation of electronics, namely tubes, made a major contribution because it was possible to manufacture them at relatively low cost. This opened up the radio market in the 1920's and 30's and was the beginning of electronics.

The transistor can be identified as defining the second generation of electronics. Reliability was its principal early contribution, with low cost soon following. A key aspect of the transistor is that it is fabricated by materials processing rather than by the mechanical techniques used to build vacuum tubes.

Integrated circuits identified a third generation of electronics; here materials processing has been expanded to where complete circuits are fabricated on a chip of silicon. Another mechanical operation, namely that of connecting together discrete devices into circuits, has been eliminated.

As we move into the fourth generation of electronics, namely that of Integrated Equipment Components (IEC's), it is clear that a key technological result will be the use of materials processing in place of the mechanical operations of interconnecting thousands of circuits. This will result in IEC's which are easy to use and should result in electronics becoming truly pervasive.

Structure of the Electronics Industry

Such a radical change in the technological base of electronics can be expected to have a dramatic impact on the structure of the electronics industry. Figure 24 shows the author's view as to what will happen to the electronics industry.

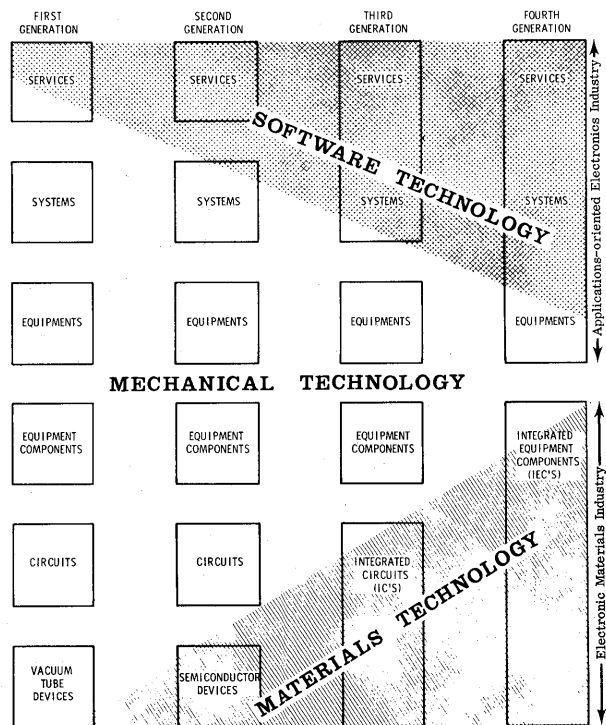


Figure 24. Structure of electronics industry.

This figure forecasts that expansion of materials technology to the IEC level will result in the integration of the device, circuit, and equipment businesses into a single business . . . segment. Those companies which will make up this business segment will integrate from materials up through equipment components. In order to do this, those companies must provide not only for materials technology, but also for device, circuit, and equipment design and fabrication capabilities. Because of the magnitude of this total problem, it seems likely that four or five large integrated⁹ electronics suppliers will provide "electronic material," i.e., IEC's, IC's and devices on a very broad base to a much larger applications-oriented electronics industry.

Because of the tremendous expansion of the use of electronics which will result from the overcoming of the four limitations of electronics discussed above, the applications side of the electronics industry will expand in an unprecedented manner. While making this expansion, we forecast in Fig. 24 that much more emphasis will be given to the "software" aspect of electronics by this applications-oriented industry. By software we do not mean simply programming, but rather use the term in the broad sense

where the customers' total problem is considered and solved.

The applications-oriented companies will use IEC's and other peripheral equipment components to provide a total solution to a customer's problem. It can be expected that this will be a very broad and diverse business consisting of many companies, a number of which are relatively small and specialize in a particular area of application. The materials-based integrated equipment component companies will relate to these applications-oriented companies much the same way as the chemical industry provides processed chemicals to a very large applications-oriented industry such as the clothing industry.

Another possible structure of the electronics industry is one that is highly integrated vertically such that systems houses provide their own IEC's. While this may happen to some degree, it is forecast that a very large materials-based integrated electronics business will still develop which supplies IEC's, IC's and devices on a very broad base to industry. The principal reason for this is the very pervasiveness that electronics will achieve. Compared to the very large number of companies which will use electronics, only relatively few companies will find it profitable to fabricate their own IEC's. Those companies whose internal requirements make it profitable for them to supply a part of their own IEC requirements will also depend upon the IEC suppliers for a significant part of their requirements. The sheer size of their needs, which is what makes it profitable for them to operate an internal facility, is also what makes it impossible for an internal operation to satisfy all of their needs.

Finally, it is forecast that the IEC suppliers will vertically integrate in selected areas of equipments, systems and services. For example, the author's own company has interest in providing geophysical equipment, systems and services to the oil and related industries. Such vertical integration will be a small fraction of the total application area of electronics.

It is concluded that the expansion of materials technology to the level where Integrated Equipment Components are achieved on slices of semiconductor will result in an electronics industry consisting of two major segments, one based on materials technology, the other based on software technology. Materials technology will provide the technological base for a concentrated Integrated Electronics industry which will supply IEC's, IC's and devices on a very broad base to a much larger application-oriented electronics

industry whose principal technology is software. Vertical integration will occur selectively, but will have no major effect on the overall division of the industry into these two major segments.

ACKNOWLEDGMENTS

The author is indebted to his colleagues at Texas Instruments for many stimulating discussions and helpful work on the subject of LSI: in particular, to Jay Lathrop for discussions of the overall aspect of the technology of LSI; to Jack Kilby for his personal leadership of the discretionary wiring approach to LSI; to Ray Warner for the forecasts and insight into the MOS technology.

With respect to more general aspects of LSI, discussions with Richard J. Hanschen, Cecil Dotson, Willis Adcock and Jack Kilby have been most helpful. And finally, the farsighted vision of Patrick E. Haggerty on the pervasiveness and general impact of Integrated Electronics has provided stimulus to the author's thinking.

REFERENCES

1. *Proc. IEEE*, vol. 52, Dec. 1964, Integrated Electronics Issue.
2. J. W. Lathrop, *Proc. IEEE*, vol. 52, pp. 1430-43 (Dec. 1964).
3. Richard L. Petritz, *ibid*, vol. 50, no. 5, pp. 1025-38 (May 1962).
4. —, *Trans. Met. Soc. AIME*, vol. 236, pp. 235-49 (1966).
5. Inter. Solid-State Circuits Conference, Feb. 9-11, 1966, Philadelphia; Western Electronics Show and Convention, Aug. 23-26, 1966, Los Angeles.
6. P. K. Weimer, *Trans. Met. Soc. AIME*, vol. 236, pp. 250-56 (1966).
7. H. C. Josephs, "A Figure of Merit for Digital Systems," *Microelectronics and Reliability*, vol. 4, pp. 345-50 (1965).
8. J. W. Lathrop, "Discretionary Wiring Approach to Large Scale Integration," Western Electronics Show and Convention, Aug. 23-26, 1966, Los Angeles.
9. P. E. Haggerty, "Integrated Electronics—A Perspective," *Proc. IEEE*, vol. 52, pp. 1400-5 (Dec. 1964).

EFFECTS OF LARGE ARRAYS ON MACHINE ORGANIZATION AND HARDWARE/SOFTWARE TRADEOFFS

L. C. Hobbs

*Hobbs Associates, Inc.
Corona del Mar, California*

INTRODUCTION

From the early days of electronic computers until the present, a period of over 20 years, electronic and magnetic hardware for mechanizing logical functions and storage in the central processor portion of a computer system have been extremely expensive. Although these costs have been dropping steadily in terms of the cost per component, increases in the complexity and capacity of central processors have tended to keep pace with decreases in hardware costs. Hence, reductions in hardware costs to date have been reflected primarily in increased performance and capability rather than reduced cost. However, developments presently underway in batch-fabricated technologies will provide such significantly lower hardware costs in the central processor that it will not be possible to maintain a system balance from the standpoint of cost and reliability. If properly used, large-scale integrated-circuit arrays in particular will provide digital logic and control functions at such sharply reduced costs and increased reliability that the central processor will tend to become an almost negligible part of the system from the standpoint of both cost and reliability. The dominant factors in systems cost will be software and electromechanical mass storage and input/output devices.

As a result of these technological advances in batch-fabricated hardware, the three major problems facing designers of future computer systems will be:

1. The necessity of developing machine organization techniques that will permit the efficient utilization of large arrays to achieve their true potential in terms of cost, reliability, and maintainability.
2. An urgent need to minimize the number of electromechanical mass storage and input/output devices required in a system in order to reduce systems cost and increase systems reliability and an accompanying need for developing new and improved types of such peripheral equipments.
3. An equally urgent need for minimizing the cost of providing software, including both operating systems and user programs, even if this requires significant increases in the logical and storage hardware in the central processor.

MACHINE ORGANIZATION IMPLICATIONS

In considering the advantages of large arrays it seems apparent that the larger the array that can be

effectively utilized the better the economics and reliability up to the limit that can be achieved technically in terms of the number of components per chip.¹ The use of very large arrays will reduce the initial fabrication costs and improve reliability and maintainability, but they will also present a serious problem. As the module becomes larger it becomes increasingly difficult to use it for more than one function within a single computer. Each packaged unit tends to become unique with only a single one of each type used in a given computer. Dr. R. N. Noyce called attention to this last year and indicated the anticipated progress in array size when he stated:

However, from a point on the complexity scale now where 50 components is the cheapest level for an integrated circuit, I expect to move to 1000 by 1970. . . . At the same time there will be new problems where it takes only 10 chips to make a computer and almost every circuit made will be different.²

This decreased "commonality" increases fabrication costs because of the low production volume of each type of module. It also increases the cost of the spares inventory, but the cost of spares usage will decrease as a result of significantly higher reliability. In fact, the low rate of usage of spares coupled with the difficulty of repairing large arrays should lead to adoption of a "throw-away" maintenance concept where major portions of the computer are replaced but not repaired in event of failure. This will have significant effects on maintenance procedures and costs—particularly in military systems.

The lack of flexibility in large arrays which tends to make each array within a system unique and the possible need for eliminating bad or substandard circuits from the array to achieve a reasonable yield are two of the major problems in utilizing large arrays in computers. At least three different approaches to fabricating large interconnected arrays to overcome these obstacles to their utilization are under consideration. The first is cellular logic in which large arrays of identical circuits are fabricated with a standard interconnection pattern (e.g., connecting each circuit only to its four adjacent neighbors) with the ability to modify the function of the circuit by changing something in the circuit subsequent to fabrication.³ For example, one approach of this type uses a circuit with four cut-points which can be cut in different combinations to alter the function of the circuit.

In the second approach, a large array of circuits

is fabricated and each circuit is individually tested. The test results are put in a computer which is also storing logical equations of the function to be implemented. The computer then generates the proper interconnection pattern to interconnect available good elements (skipping the bad ones) to perform the required logical function.⁴ In this approach, a separate mask must be prepared for each array fabricated; hence, this is an expensive operation unless cheap methods can be developed for producing interconnection masks under computer control. On the other hand this approach offers a major advantage in that it is easy to vary the function performed by the array by changing the logical equations supplied to the computer that is controlling the interconnections. If each interconnection mask for each array is generated individually, there is little incentive for rigidly standardized functions.

The third approach is advocated by those who believe that in the future it will be technically feasible to achieve high yields of large integrated circuit arrays in which all circuits are good. This would permit a standardized interconnect pattern to be used for each specific logical function. This has the advantage that only one mask need be made for a particular function. This mask can then be used to interconnect the circuits in many arrays of that type. On the other hand, it is more difficult to change the function to be performed by the interconnected circuit array since this requires making a different mask.

In the future both of the last two fabrication techniques discussed above will probably be used. Programmed control of the interconnection pattern will likely be used for small production volumes and unique or infrequently used functional modules. However, there is strong evidence that the semiconductor industry will produce large arrays with yields sufficiently high to permit the use of standardized interconnection patterns for functional modules that are used in large quantities.

As semiconductor and batch-fabrication technologies advance, the major physical limitation on the size of the functional unit will be the number of external leads that can be provided on a package. Although packages with larger numbers of leads (in the order of 100) are being developed, additional research in machine organization is urgently needed to develop functional organizational concepts that will maximize the interconnections within a replaceable package and minimize the interconnections be-

tween packages. The way in which the computer is divided into functional modules can greatly increase or decrease the number of connections needed between such modules.⁵

It will be necessary to use different criteria for design efficiency in batch-fabricated systems. In the past, minimizing the number of logical elements has been a major goal of most logical design efforts. In future systems, logical elements should be used inefficiently in order to minimize the number of interconnections needed between functional modules. For example, frequently in present computers a given gate or flip-flop supplies inputs to a number of logical elements in different parts of the machine; but in future systems the logical gate or flip-flop may be duplicated many times in different parts of the system to minimize the signals transferred from one module to another. Emphasis must be placed on reducing the number of packages and the number of interconnections between packages—even at the expense of increasing the logical complexity of each package significantly. Perhaps an even more difficult problem will be motivating logical designers and systems planners to use standard or predetermined functional modules. It is hoped that Dr. E. A. Sack's optimism was justified when he stated: "It is the author's opinion that the drastic reduction in cost per gate available in multi-gate arrays will overcome the system designer's natural reluctance to employ *prefabricated* digital functions."⁶

In order to achieve the cost and maintenance advantages offered by the use of large batch-fabricated arrays, it is necessary to develop machine organization and system design techniques that permit repetitive use of packages containing very large arrays of circuits. One approach is to change the internal organization and logical design of the large computer so that large functional arrays can be used repetitively even if this means that each array is relatively inefficient in terms of the utilization of circuits within the array.⁷

Another approach is to use very small standard modular computers designed to be used either individually or in multicomputer systems. In this case, the uniqueness of large functional arrays within a given computer is accepted. Such a small standard modular computer can be fabricated with a very limited number of circuit arrays each of which is used only once within that computer. For example, the complete program control unit and all of its internal interconnections may be fabricated in a single

package, the complete arithmetic unit in a second package, the complete input/output control and buffering section in a third package, and storage modules in additional packages containing 2000 words each. Economies in fabrication and spares inventory would be achieved as a result of the volume usage of each type of module made possible by the use of a large number of these standardized computers rather than by the use of a large number of identical packages within a single computer. When additional computing speed and capability is required the standardized computer would be used in a multi-computer configuration.

A third approach is to develop parallel processing systems conceptually similar to those that have been discussed extensively in the literature.^{8,9} In this approach, large arrays are used effectively by organizing the machine on the basis of a relatively large number of identical processing modules.

INPUT/OUTPUT IMBALANCE

There are three major approaches to the input/output problem:

1. Improvements in the performance of present types of input/output equipment.
2. Development of new types of input/output equipment that are not in widespread use at present.
3. System organization approaches that minimize the need for conventional input/output equipment.

Each of these approaches will play a part in providing better balance in future systems. However, unless much greater effort is placed upon the development of nonmechanical input/output equipment, the best hope for future systems probably lies in developing system techniques that minimize the need for input/output equipment. Although a problem of major importance, these have been discussed previously and will not be considered further here.¹⁰

HARDWARE/SOFTWARE TRADEOFFS

The memory capacity of early computers was so limited that programming costs were not a significant part of the total cost-of-ownership of a computer system. However, reductions in hardware costs have been accompanied by greatly increased memory capacities which have permitted the storage

and operation of larger and more complex programs. The cost of hardware and the cost of engineering design required to efficiently use expensive logical components have exerted a strong pressure in the direction of very general-purpose computers which can be adapted to a large number of different operations so that design and production costs can be amortized over a relatively larger number of units. It has been recognized that a special-purpose computer can perform a particular task more efficiently than a general-purpose computer in terms of the amount of hardware required, but the cost of small volume production and specialized design have favored general-purpose computers.

Under these circumstances, the tasks of specializing the capabilities of a general-purpose computer to a specific job and adapting it to the control of a large number of different types of input/output and peripheral devices have been left to the programmer. However, the increased performance and capability of computers that have accompanied the reductions in basic hardware costs in recent years have placed greater and greater requirements on the programming necessary to adapt more sophisticated general-purpose machines to more complex operations in specific kinds of problems.

While hardware costs have been decreasing, programming costs have been increasing significantly to the point that they now represent at least 50% of the initial cost of a new computer system and perhaps as much as 80% of the total systems operational cost over a 10-year period. This problem is now magnified by new batch-fabrication technologies, such as large-scale integrated circuits and plated-wire or thin film memories, which are expected to reduce the cost of logic circuits and storage elements by one to two orders of magnitude. However, the effect of these hardware cost reductions on the cost-of-ownership (initial procurement cost and systems operational cost over the lifetime of the system) is limited by the overwhelming software costs which will not be affected by these advances in hardware technology unless machine organization and system design concepts are changed.

Fortunately, the significant reductions that are being achieved in the cost of logic and storage offer an opportunity to also reduce the mounting cost of software by trading low-cost hardware for expensive software. Many of the functions relegated to programming in the past because of high hardware costs can be performed in the future by low-cost

batch-fabricated hardware with a consequent reduction in programming complexity and costs. Technological changes now make it necessary to reverse the past practice of using additional software to minimize hardware requirements. In the future, additional hardware will be used to reduce programming requirements. This can only be achieved by reevaluating the criteria used for making hardware/software tradeoffs in the past.

At least three different approaches to altering previously accepted hardware/software tradeoffs can be considered:

1. Special-purpose computers and processors.
2. Different types of machine language and machine organization.
3. Additional hardware functions in machines with conventional languages and organizations.

Special-Purpose Computers and Processors

The question of special-purpose versus general-purpose computers has been argued in one way or another since the dawn of the computer era. The major arguments against special-purpose computers have been design costs and lack of flexibility. Special-purpose computers have been frequently favored for applications where a relatively large number of machines have been required to do a certain set of fixed tasks, but in most such cases some limited form of program control (e.g., paper tape, plug board, etc.) has been added to provide some flexibility. With the advent of computer-aided design and computer-controlled preparation of masks for large-scale integrated circuits much of the design cost obstacle is removed. In essence the question then becomes one of trading logical design in a special-purpose machine for programming in a general-purpose machine. In this case, the logical design will probably win out in terms of the number of man-hours required since the logical designer can address himself to the task at hand with few predesign boundary conditions while the programmer does not have a completely free hand because of the characteristics of the general-purpose machine he is adapting to a specific problem.

The problem of flexibility remains, but this may be partially overcome by a compromise in a multi-computer or multiprocessor system. A discussion of the many advantages of multicomputer and multi-

processor systems is outside the scope of this paper, but in many cases it is not necessary that all of the processors or computers in such a system be identical nor that they all be general-purpose. A multi-computer or multiprocessor system is feasible in which some of the computers or processors are general purpose while others are special purpose, designed to perform specific tasks that are relatively common. For example, in a multicomputer scientific computation system one or more of the computers could be DDA's. As another example, in a multiprocessor system one of the processors could be a logical processor, another an arithmetic processor, etc. It seems fairly obvious that such use of special-purpose computers or processors will reduce the programming requirements (as well as probably increasing processing speeds) and will be economically feasible when the low-cost potentials of large-scale integration are realized.

Different Types of Machine Language and Organization

Present machine languages and machine organization concepts have been heavily influenced by the cost and capabilities of specific types of hardware in the past. The storage hierarchy is one example of this. The sequential one-address machine language is another. If word size were not limited by the cost of larger registers and storage, three-address machines would undoubtedly be more prevalent, particularly in data processing type applications.

Machine languages have been designed to permit efficient implementation of the processor itself rather than to facilitate programming. Users on the other hand have developed pseudo-languages that facilitate programming from a human standpoint but that require compiling operations that do not always utilize the true capabilities of the computer. On the surface there seems to be an advantage in using some higher order languages (e.g., FORTRAN or ALGOL) as machine languages, if hardware costs are sufficiently low. It will probably not be feasible to go this far, nor is it necessarily desirable. However, it is feasible and desirable to design machine languages that will facilitate compiling operations and to implement certain parts of problem oriented languages in hardware. The need for better problem oriented languages has been cited frequently.¹¹ Hence, a joint effort by programmers and engineers to first design better problem-oriented languages and then to implement portions of them (e.g., mathematical

operations) with hardware wherever possible should pay handsome dividends.

The Burroughs B5000 with its Polish notation and push-down-list store represented an early step toward development of machine languages and organizations that will facilitate compiling operations.¹² Other work in this direction includes an Air Force-sponsored study at the University of Pennsylvania using associative memory and list processing techniques.¹³ With very low-cost large-scale integrated-circuit arrays just over the horizon, it should be economically feasible to implement machine languages that will eliminate many of the steps in present compiling operations.

Additional Hardware Functions in Conventional Machines

It is not necessary to go as far as special-purpose computers or new machine languages and organization to achieve significant economies in software by greater, and perhaps "inefficient," use of low-cost hardware. Significant economies can be achieved within the framework of conventional machine languages and organizations by:

- Hardware implementation of special purpose functions and logical and mathematical operations.
- Implementation of hardware features that minimize "red tape" and "house-keeping" programming requirements.
- Hardware implementation of some of the machine functions presently handled by operating systems software.

Many functions presently handled by programmed subroutines can be implemented easily by special-purpose logic in a straightforward manner. Such functions include:

- Binary-to-decimal and decimal-to-binary conversions
- Code conversions
- Coordinate conversions
- Format control
- Table look-up operations
- Scaling
- Mathematical operations (e.g., square root, trigonometric functions, matrix operations, etc.)

In the past such functions have been handled by programmed subroutines using the machine's basic

operations (e.g., add, multiply, shift, etc.) because of the cost of the hardware required to mechanize the functions in relation to the frequency of their use and because of the flexibility offered by the ability to modify the routine either as it is stored or by index registers at the time of execution. From the cost standpoint, large-scale integration will make it feasible to mechanize such functions even when they are used relatively infrequently. The necessary flexibility can be retained by hardware mechanizations that permit program control of variable operations in such functions and still significantly reduce the software required. Hardware mechanization of functions of this type will not only reduce the programming effort and the storage space required for the program, but will also offer speed improvements since logical implementation of such functions is invariably faster than the execution of the equivalent sequence of program steps.

A large portion of most programs consist of "red tape" or "housekeeping" instructions that either are not conceptually necessary to the solution of the problem or that can be implicit to the operation performed rather than stated explicitly. These include operations such as:

- Register-to-memory or memory-to-register transfers
- Certain transfer-of-control operations
- Some operations on the contents of index registers
- Certain types of timing functions

Additional hardware can greatly minimize the number of operations of this type required in a program. For example, a set of general registers or a small high-speed control memory can be used to represent multiple accumulators, index registers, and control registers. The availability of such multiple registers will sharply reduce the number of register-to-memory and memory-to-register transfers, the number of index modification operations, and the number of transfer-of-control operations required. There are, of course, many other examples of this type. A somewhat complementary concept is the use of large-capacity low-cost storage coupled with higher-speed machine operation to permit the effective utilization of inefficient programs. This increases the size of the program in terms of the number of instructions involved but reduces the man hours required to write a given program by removing the need for polishing and streamlining the program to

make it run faster and fit into less storage space.¹¹

The operating systems software provided with most computers handles three major functions:

- Input/output control and editing
- Scheduling and storage allocation
- Interrupts and priorities

The operating systems usually represent the most difficult and expensive area of systems programming. It has been estimated that one major computer manufacturer is spending \$60 million this year for programming PL1, FORTRAN, COBOL, and the operating systems for a family of new computer systems. The operating systems probably represent at least one half of this cost.

Many of the functions included in operating systems software can be implemented by additional hardware. For example, special-purpose control logic and storage hardware can be provided with each type of input/output equipment to provide a completely standard interface with the computer so that the programmer and the software system need not be concerned with the nature or characteristics of the particular input/output device. Special-purpose hardware and buffer storage can accommodate the differences in characteristics of tape units, disc files, card readers, keyboards, etc. Small associative memories can be used to facilitate the cataloging and indexing of data files and the allocation of storage. Hardware can significantly reduce programming requirements in the servicing of interrupts and handling of priorities. Computer and I/O channel usage accounting can be facilitated by additional hardware.

Most present computer systems use a multilevel storage hierarchy which usually requires program consideration of the particular level of storage being used and to some extent the differences in the characteristics of devices used for different levels. Additional low-cost logic and special storage techniques can be used to cause this multilevel storage hierarchy to appear as a single homogeneous storage to the programmer, thus minimizing the need for programming attention to the capabilities and characteristics of the different types of storage. Many of the storage allocation, page turning, and memory protection schemes used for time-sharing, multiprogramming, multiprocessor, and multicomputer systems can be implemented by low-cost hardware also.

FUTURE PROGRESS

Special-purpose computers or processors may evolve naturally as multicomputer and multiprocessor systems are developed and used on an increasing scale. Hardware features to minimize housekeeping will also tend to evolve as designers find lower and lower cost elements and functions available to them. The hardware implementation of special-purpose functions is straightforward, but a catalog of present subroutines can give a clue to the functions to be considered for implementation.

Present compilers and higher-order languages are a good starting point for considering machine languages and organizations that will simplify programming; but, even with low-cost hardware, further research in programming languages is needed to determine a language closely related to users' problem oriented languages that is still economically and conceptually feasible to implement as machine language.

Much of the conceptual work necessary to implement operating systems functions has already been done. The large and complex software operating systems that have been developed during the past 8 to 10 years represent many man-years of effort in formalizing the necessary procedures and algorithms. Hence, the starting point should be a study of these operating systems to determine areas that meet three criteria—(1) difficult or unsolved problems, (2) significant numbers of instructions, and (3) procedures and algorithms that are more feasible for mechanization by large-scale integrated circuits or other batch-fabricated hardware. Software functions meeting any of these criteria represent a promising area for development. Functions meeting all three will literally represent a gold mine of software cost savings.

Joint hardware, software, and systems design efforts are needed in choosing new hardware/software tradeoffs to properly utilize both the results of past work and the capabilities of new technology. In the past, hardware has been traded for software in order to improve speed and performance with the decisions made primarily on a cost/performance basis. In the future, hardware will be traded for software to reduce the cost of programming with decisions made on the basis of total systems cost rather than only equipment costs.

When transistors were first introduced there was a strong tendency to use them in the same way vacu-

um tubes had been used previously. Unfortunately in the computer field a similar trend is in process now with integrated circuits being used in the same way that discrete semiconductors have been used in the past. Systems designers must consider large-scale integrated-circuit arrays as a new type of device that necessitates major revisions in systems design concepts, machine organization, and hardware/software tradeoffs.

ACKNOWLEDGMENTS

The author would like to express his appreciation and acknowledge the assistance of several people with whom the subject of this paper was discussed—particularly D. R. Ream, R. Rice, T. B. Steel, Jr., R. G. Tuttle, and D. F. Weinberg. Many of the ideas covered in this paper are an outgrowth of material developed during a study of "Technology Applications for Tactical Data Systems," sponsored by the Naval Analysis Group of the Office of Naval Research under NONT-4910(00).

REFERENCES

1. E. A. Sack, R. C. Lyman and G. Y. Chang, "Evolution of the Concept of a Computer on a Slice," *Proceedings of the IEEE*, vol. 52, no. 12, pp. 1713-20 (Dec. 1964).
2. R. N. Noyce, San Diego Council of WEMA.
3. R. C. Minnick, "Application of Cellular Logic to the Design of Monolithic Digital Systems," *Microelectronics and Large Systems*, Spartan Books, Washington, D.C., 1965, pp. 225-47.
4. J. S. Kilby, "Device Fabrication," *Proceedings of the 1966 International Solid-State Circuits Conference*, p. 30.
5. R. Rice, "Systematic Procedures for Digital System Realization from Logic Design to Production," *Proceedings of the IEEE*, vol. 52, no. 12, pp. 1691-1702 (Dec. 1964).
6. E. A. Sack, "Complex Digital Integrated Circuits: An Opportunity for the Logic Designer," *Microelectronics and Large Systems*, Spartan Books, Washington, D.C., 1965, pp. 141-54.
7. R. Rice, "Integrations—The Predictable Effects on Engineering," *Proceedings of the National Symposium on the Impact of Batch Fabrication on Future Computers*, pp. 237-53.
8. J. H. Holland, "Iterative Circuit Computers: Characterization and Resume of Advantages and

Disadvantages," *Microelectronics and Large Systems*, Spartan Books, Washington, D.C., 1965, pp. 171-78.

9. G. A. Crane, "Economics of the DDLM, A Batch-Fabricatable Parallel Processor," *Proceedings of the National Symposium on the Impact of Batch Fabrication on Future Computers*, pp. 144-49.

10. L. C. Hobbs, "The Impact of Hardware in the 1970's," *Datamation*, vol. 12, no. 3, (March 1966) pp. 36-44.

11. T. B. Steel, Jr., "Promising Avenues of Research and Development—Programming Research," panel discussion at 1965 FJCC.

12. "The Descriptor, A Definition of the B5000 Information Processing System," Burroughs Corporation, Detroit, Mich., 1961.

13. H. J. Gray et al, "Interactions of Computer Language and Machine Design," Technical Report No. RADC-TR-64-511 (AD617-616), University of Pennsylvania (May 1965).

A PROSPECTUS ON INTEGRATED ELECTRONICS AND COMPUTER ARCHITECTURE

Michael J. Flynn

*Northwestern University
Evanston, Illinois*

INTRODUCTION

In order to assess the impact of any technological breakthroughs on the computer user, it is first necessary to understand the nature of the computer user's problem in its present context. Indeed, it is a source of considerable disappointment to some that radical improvements in technology over the past several decades have resulted in noncommensurate impact on the user. The purpose of this paper is to examine the question: "What will be the impact of integrated circuits or integrated electronics on computer architecture?" In the first part of the paper we will study the present picture of computing and show that, if no changes are made from present operating procedures, the impact will be small. In the second part of the paper, we analyze several aspects of computing to find conditions which will enlarge the impact. The final part of the paper will list a number of open problems in computing for which integrated electronics may provide a solution.

PRESENT STATUS

To begin with we define the term architecture, a term first used in this context by F. P. Brooks, Jr.¹ Computer architecture is the structure of the process that solves a user's problem. It cannot be regarded as simply a study of a processor itself but rather as

a disciplinary concern for the entire environment of the user. This process is much more than one machine or one piece of apparatus. It is, in fact, a whole ensemble of equipment, facilities, people, arithmetic tools, linguistic skills, and economic conditions.

In order to understand the computer user's problem, we must introduce a measure of effectiveness. The measure I have chosen is simply cost per computation (for a standard problem). We are first going to determine the fraction of computing cost represented by processor rental, and then determine the fraction of processor rental attributable to circuitry. In any given installation computation cost is directly related to the annual budget. Expense items in a budget usually include: (1) equipment rental; (2) systems programming; (3) applications consulting; (4) training of users; (5) supplies, including paper and forms; (6) general overhead, building, etc.; (7) nonprofessional personnel, including key-punch operators, machine-room operators; and (8) management. Figure 1 seems to be a fairly representative picture of the budget of several computing centers. Even though the numbers in Fig. 1 are not precise, the processor costs (the area traditionally associated with integrated circuitry) rarely get above 10% of the total budget. The costs described in Fig. 1 are still frequently a small percentage of the true cost of the solution of a user's problem. Neglected

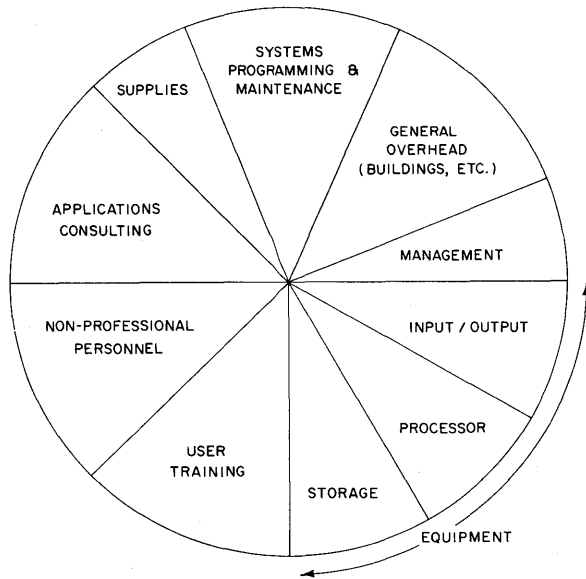


Figure 1. User's costs.

are the many program reruns for debugging, numerical difficulties and program restructuring due to improved problem insight.

Circuitry costs represent a similarly small fraction of the processor costs. The manufacturer's costs (Fig. 2) must reflect programming systems, servicing, maintenance, marketing costs, personnel overhead (which includes support activities and management), basic research and general overhead (facilities, etc.) items, as well as engineering design, records, maintenance, and variable manufacturing costs. The variable manufacturing costs include the circuitry, first-level packaging and assembly, second-level packaging and wiring, power supplies, frames, covers, cables and mechanical hardware, third-level assembly, and manufacturing debugging.

The last breakdown is open to the comment that the manufacturer's costs are not directly related to the price the user pays. This further diminishes the percentage of computing costs represented by circuitry. Thus, if circuitry costs zero, the user's cost of computing would not be affected noticeably. The basic assumption, of course, is that computing systems would be built in much the same way as they have been.

THE ARCHITECTURAL PROBLEM

In order to avoid the pitfalls of the above assumption, we reexamine the problem.

There are three aspects of the architectural problem:

1. The external, man-system communications problem.
2. The internal, system-system communications and processing problem.
3. The external, problem-system interface problem.

The Man-Systems Problem

The fact that the decision-making electronic components represent only a very small fraction of the user cost is an indication that the interface between these components and the ultimate user, man, is a poor one or at least an inefficient one.

The few percent of total cost represented by circuitry is the cost component that actually solves the problem. The remaining costs allow man to communicate with this circuitry.

This communications interface between man and machine is characterized by the following aspects:

1. Bandwidth: total amount of information provided by the machine per unit time.
2. Storage.
3. Language.
4. Responsive interaction between man and machine.

Bandwidth, per se, is not a significant problem if the storage, language, and responsive interaction problems can be solved. Presently typical problems

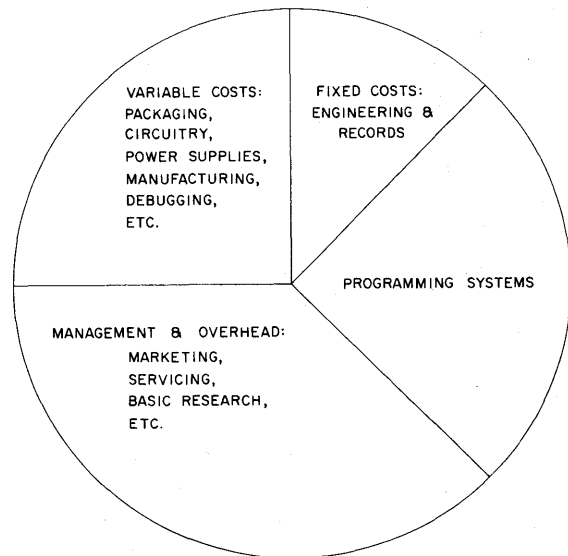


Figure 2. Manufacturer's costs of a processor.

run hundreds of pages of output. Clearly, much of the output is useless. The cause of the excess is that the human cannot interact readily with the system. The time lags are too great between the original query, the time the machine responds, and the still later time when the human can return to the system for further queries. Thus, the human demands exhaustive amounts of output data to make up for the lack of interaction.

Storage must be available on a semipermanent medium in such a fashion that it can be reanalyzed or acted on at a later, more convenient moment. The paper medium, presently used, has low cost, permanence and legibility. Its disadvantages are that printout is a relatively slow process and that the result is not easily machine-readable. Any new interface must compete with its advantages as well as solve its problems.

The third aspect of the problem is language. For various reasons, the computer as a processor does not work at the same linguistic level or in the same semantic mode that the human does. The human must adapt himself, therefore, to learn highly formalized and stylized languages. This presents a barrier to the interface since the burden is on man. As pointed out in one of the other papers in this session,² tradeoffs can be made to some degree to allow more ready linguistic interplay between man and the machine.

Attempts to implement more responsive interaction between man and machine have usually taken the form of time-sharing systems^{3,4} or "the utility concept." These user-shared systems are not the most efficient computing mechanism (due to the overhead of swapping storage and control between users). However, they do represent a higher level of man-system efficiency which, as we have discussed, is the more significant. While integrated circuitry may play a role in increasing the internal efficiency of the user-shared processor (via buffering) in this section we will consider the device with which man communicates: the display. The immediate access display is a necessary condition for the implementation of a responsive system.

The display itself has developed in phases. Early displays were unidirectional, acting as output-only rather than I/O devices. Their use was mainly for data compression or formatting (as in curve plotting). Later, semiresponsive displays were developed where interaction was restricted to manual keys or paper tape. More recently, the availability of the so-

called light-pen gives a new dimension to the interaction, with an immediate program interrupt to focus on any point of the display. In order to satisfy the aforementioned man-machine requirements, there should be another phase of development characterized by:

1. *Interactive Display*—The continued use of the light-pen with refinements in electronics would be suitable. The major problem is the development of sufficiently versatile programming languages directed at graphic, interactive devices.

2. *Solid-State Display*—The electrical mismatch between cathode-ray tube displays and integrated circuitry is an unfortunate cost impediment. A low-voltage, two-dimensional, variable-size solid-state display would be a major breakthrough.

3. *Storage*—A truly useful display should have facility for large quantity of storage which is *machine-readable* and *console-readable*. A photographic medium is an example of a low-cost, permanent storage, suitable for such purposes as data reanalysis, program storage, etc. Current technology allows the resolution of 10^6 to 10^8 bits on a 2×2 or 70mm film slide without involving mechanical motion. Photography can fulfill both of the basic functions of storage: the final (output) document, which needs only to be man-readable, and the interactive, temporary results, which should be both machine and man (via console) readable.

4. *Adaptability*—A versatile console must be usable in conjunction with any standard communication link (telephone, etc.)—i.e. it must be proximate to the user.

5. *Cost*—The overall cost should not exceed key-punch cost. It is here that integrated circuits could play the key role by providing complex functions at reasonable cost.

Displays, available or under development, already exhibit some of these characteristics in varying degrees. A display which fulfilled them completely would have great architectural significance. It would represent (from the user's viewpoint) the elimination of the input/output factors, the keypunch personnel, the endless program re-runs, and in general the wider acceptance of the computing system. Indeed, it would constitute an order-of-magnitude breakthrough for the user.

While consoles, as above, could be associated with either small, private computers or large, shared

“utility” computers, I feel that the latter will offer the general user a much more attractive interface. By interface, I mean libraries of programming systems and subroutines, the availability of human consultants for difficult problems, and the availability of, or remote access to, very large data files.

In spite of the obvious importance of the display area, quantitative data on relevant aspects of human learning with respect to the computing process are scarce. And while an encouraging amount of interest has been shown by manufacturers, too often the context of their interest is gimmickery, rather than studied usefulness.

Systems—Systems Communications and Processing Problem

Within the system, communications is an electrical signal processing problem and has been widely discussed. At least within the processor, large monolithic arrays will be the basic decision subunit of the system. Storage and I/O storage functions pose a difficult problem. Storage is usually regarded as a hierarchy in which the most immediate element, the main storage, has fast access with limited amounts of storage capabilities. Higher up on the hierarchy are the drum and the disk, which have substantial capacities, but have latencies of several milliseconds to access the first piece of data. Once access is achieved, transfer is usually made at rates substantially less (a factor of 10) than the main storage rate. This mismatch and the mismatches at higher hierarchical levels such as tape and I/O represent a key systems problem.

While integrated circuitry holds promise for some implementations of main memory, it is still prohibitively expensive to consider as a storage hierarchy replacement. The only hope, it would seem to me, is the use of integrated circuits in conjunction with an inherently high-density storage medium. Developments along this line would be most welcome.

With respect to the processor and main storage, use of monolithic arrays has two important implications: (1) Logic design must be viewed as an insular problem wherein substantial amounts of circuitry are connected on a single chip; the criterion which determines the size of these logical islands is the number of external connections which are made to it. This requires that optimum use of connection bandwidth be made between units; otherwise, cost premiums will exist due to first and second level

packages. Also, for similar speeds, the communication to an external unit demands substantially more power than to an internal circuit decision element. Hence, to make most efficient use of power, the minimum number of external connections should be made. Thus one would tend to make maximum use of external connections or the bandwidth of the external connections. (2) The most significant factor in costs is not the number of circuits per chip, or even by the total number of chips employed, but rather the number of different kinds of chips.

Thus, the problem reduces itself to one of partitioning the system into a set of logical islands, each with a minimum number of external interconnections which involve the minimum number of different type of islands. We will examine main storage, controls, data paths, arithmetic, and general logic design considerations as the five fundamental areas of the system to determine the influence of these constraints.

1. *Main Storage.* The nature of storage is such that the connectivity and replication constraints are always satisfied. The number of input/output pins increase linearly with the logarithm of the number of bits contained on the chip. The problem of main memory is not these initial constraints but rather the economic advantage of competitively established technology, such as magnetic core or film, which have very efficient methods of interconnection and data storage. For integrated circuits to provide a valuable storage function, one must take advantage of the availability of its logical power. This allows consideration of such storage arrangements as the associative or content-address memory. In the associative memory, a known subportion of a data word is presented to the memory and inquiry is made on the appropriate portion of all such words contained in that storage. If match is detected, it is indicated, and the remaining (associated) information contained in the word is retrieved. With such memory, for example, sorting of data is no longer required. The cell structure of the associative memory is repetitive, and with the absence of address inputs the cells require less interconnection than a corresponding array of coordinate-addressed storage.

2. *Controls.* A traditional implementation of the control function is difficult if not impossible to duplicate on the basis of array-replicated circuitry. Thus it is most natural to consider micro-programming for this function. Here storage plays the role of

combinatorial and sequential decoder. The instruction represents merely the address of the initial element in the sequence of gating descriptions which are retrieved and acted upon. The micro-program arrangement is much more versatile than the traditional implementations. The fact that it is reloadable makes possible rearrangements of major portions of the machine and introduction of new instruction repertoires. Most micro-program machines to date have been of the read-only variety, where changes in the micro-program storage is an involved task. There is no reason why, with availability of integrated circuits, a machine could not dynamically restructure itself by changing the contents of its micro-program storage.

3. *Data Paths.* The principal functions of a data path are transfer of information from one subunit to another and provision of temporary storage at these entry and exit points. The data path then is a communications medium for logical units or subunits of the system. Data path circuitry may well be thought of as integral to the logical unit with which it is associated, thus insulating the unit from the outside environment. Further discussion on the external communications implications of the data paths will be made below.

4. *Arithmetic.* Arithmetic has equipment implications because it involves communications not only from the data paths but also across the numerical field of the word operated on. In selecting an algorithm for add, multiply, or divide, one must take care to maximize the efficiency of the implementation; for example, shifts could be accomplished on a digit rather than a bit basis, thus simplifying the number of interconnections for multiplying.

5. *Considerations in Logical Design Using Integrated Circuits.* As with the intra-system problems, the logic design problems can be divided into two groups: the internal problems, and the external problems of communication between modules:

- (a) *Internal logical design.* Decision elements internal to a monolithic array have an ideal environment, with excellent tracking of component values and power supply tolerances. Logical structures, such as multivalued logical structures, threshold logic gates, and specialized current summers, which were marginal at best in discrete im-

plementations, now become realizable. Respectable speeds may be maintained at the expenditure of much less power than the externally directed circuit. Thus the only problem is to keep as many logical nodes as possible free from the restrictions of the outside environment.

- (b) *External logical design.* On a typical small card of about 25 square inches, fully populated with "flat packs", less than one per cent of the area is taken up by decision-making logical components; fully 99% of the area is devoted to leads and to connections. Power is also a problem, but power is normally associated with driving the stray connector and line capacities.

It is also interesting to note that present communication techniques are very inefficient. For example, it is rare that any line passes more than 50×10^6 bits per second; whereas the capacity of that line exceeds 10^9 bits per second. This would be true for most terminated transmission lines. To relieve the communication-interconnection problems, it would be sufficient to serialize the data and maximize the transmission rate. Of course this introduces complexity in the array, because now the information must be commutated; but since we have presumed that the array originally had substantial amounts of logical decision capabilities this complexity would be acceptable.

Thus we may see the partial return to the old serial computer where we have one input line, one output line operating at very high speeds, and processing done within the array at lower speeds and in parallel.

Problem-System Interface

Thus far, we have discussed the potential for integrated electronics in the man-system relationship and in intrasystem optimization. There is a much broader level of application of integrated electronics: the direct problem-system implementation, with man eliminated as intermediary. "Closing the loop" has been the preserve of analog computers due to cost factors. Note that it is exactly in such a "people free" environment that the cost potential of integrated electronics is maximized. But even this misses the

singular promise of integrated electronics in the development of low-cost, versatile, high-performance digital transducers. This "digitizing the universe" (environment) is one of the truly expansive markets of the future.

OPEN PROBLEMS

Over the past decade, in device areas, logic design, and systems work, we have optimized our components along several dimensions: ultrahigh-speed computing, minimum-cost computing, or cost-performance computing. I believe many of our problems lie outside these broad classifications, and the pursuit of these open problems would provide substantial "fallout," namely a much greater understanding to the entire computing process. They represent overlooked dimensions of computing. In addition to man-machine interactive computing, previously discussed, some of these areas might be:

1. *The Ultra-Reliable Computer.* The problem is to build a computer whose failure rate is arbitrarily small. It is replaced rather than field-serviced. It should be able to operate in spite of any single failure, and there should be some confidence of operating over a large class of multiple failures. In order to take intelligent steps toward the realization of such a system, reexamination of many areas is required: device failures, logic design techniques, coding techniques, diagnostics, new concepts in input/output equipment, and many others. It is clear that much more reliable systems can be built than the traditional triple-modular redundancy systems; but even more significant than the resulting system itself would be the improved understanding of failure mechanisms in general.

2. *Physical Environment Area.* There would be numerous advantages in a computer which could operate under the most extreme environmental conditions and/or use the lowest possible power. Here again, this is a problem of more than just a device; it is a problem in circuit design and in logic and system techniques.

3. *Electronic Library.* The third area is the problem of the electronic library. This involves the construction of a very large (about 10^{10} bits) electronic storage with any bit accessible in under 10 microseconds. In present systems such arrangements must be handled, in part, by electromechanical storage media, and random accessing must be in the order of milliseconds to seconds; thus many problems

which have requirement for very large random memories (with interaction among any and all data in the memory) take excessively long times to be performed. The existence of such an electronic library would present a three or four order-of-magnitude breakthrough for these problems. To be most useful, such a memory should be operable in either the associative or coordinate address mode.

4. *The One-Chip Computer.* The one-chip computer has been long discussed by Holland⁵ and others. While each chip is a complete computer, its nature would be computationally atomic. The utility of the one-chip computer is in the possibility of assembling many chips into a vast network, the size of the network determining the performance of the total processor. Thus, the processor may be expanded to suit any computational need.

5. *The Maximum Efficiency Computer.* It is recognized that most general-purpose computers are general-purpose only in the sense that they are equally inconvenient for all to use. It should be possible to design a system whose organization is flexible and versatile enough so that it might undergo a maximum restructuring by each user in a dynamic fashion. Thus, it might employ the micro-programming techniques previously discussed and incorporate them into a versatile arrangement of data paths and logical subunits. The user would first describe the machine he would like to have work on his problem, and essentially load the instruction repertoire that he desires into this microinstruction storage. He then performs his program on the computer that he has designed to be optimum for the solution of his problem.

I feel that these problem areas have merit in themselves—as much as very-high-speed computing—and that they warrant the attention and interest normally associated with the more familiar computing dimensions. They would not necessarily be the fastest and/or the most economical system; rather hopefully, they would be useful new horizons in computing.

CONCLUSIONS

We have seen that the present interface between man and machine is inadequate, and properly designed displays may provide a significant part of the solution.

Within the framework of a computing system, integrated electronics is equivalent to monolithic

integrated circuits. There appear to be two important considerations for the logic designer: maximizing the number of decision elements per interconnection, and minimizing the number of different array types used.

On a broader level, the more removed from the system man becomes, the more the potential of the integrated electronics can be realized.

The message of this paper is a simple one. It is that integrated electronics should be much more than monolithic circuitry. The entire realm of physical resources, integrated with the systems resources we call computer architecture, should be directed toward furthering the best possible solution of the computer user's problems. The traditional form of these problems frequently masks their essential nature. It is only when this essential nature is understood that optimum technologic implementations are possible.

REFERENCES

1. G. M. Amdahl, G. A. Blaauw, and F. P. Brooks, Jr., "Architecture of the IBM System 1360," *IBM Journal of Research Development*, vol. 8, no. 2, pp. 87-101 (Apr. 1964).
2. L. C. Hobbs, "Effects of Large Arrays on Machine Organization and Hardware/Software Tradeoffs," this volume.
3. F. J. Corbató and V. A. Vysseotsky, "Introduction and Overview of the Multico System," *Fall Joint Computer Conference*, Vol. 27, Spartan Books, Washington, D.C., 1965.
4. W. T. Comfort, "Computing System Design for User Service," *ibid.*
5. J. H. Holland, "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously," *Eastern Joint Computer Conference*, 1959.

THE SYSTEM/SEMICONDUCTOR INTERFACE WITH COMPLEX INTEGRATED CIRCUITS

Wendell B. Sander

*Fairchild Semiconductor, Research and Development
Laboratory
Palo Alto, California*

INTRODUCTION

Batch-fabrication techniques in thin film, cryogenic and semiconductor technologies have been recognized for some time as potentially having a dramatic impact in computing systems, but the problem of how to translate batch-fabrication technology to advanced systems has been brought home by the recent flurry of activity in high complexity integrated circuits. In the classic tradition of keeping all existing computers obsolete, the emergence of orders of magnitude increase in device complexity comes just as the earliest conventional integrated circuit computers are available.

In attempting to discuss the system/semiconductor interface problems with devices of over 100 gates of logic power, it is interesting that no clear-cut single interface exists with conventional integrated circuits. The decision to buy standard, design custom, or build-your-own is still a subject of long debates.

The difference between conventional integrated circuits (IC's) and complex integrated circuits is basically in logic interconnection. Conventional IC's can be represented by simple logic expressions, (multiple NAND gates, flip-flops, etc.), whereas the complex circuits are some interconnection of the simple logic blocks (counters, shift registers, adders,

etc.). Since circuit design and logic design have been traditionally separated, it is fairly easy to select a conventional IC family and present it to the logic designer with the same kind of rules he had before. Conversely, it was a relatively easy task for the IC manufacturer to come up with a realistic IC family, since discrete circuit families of comparable complexity had been used for years. The circuits might change but the logic function required was fairly clear-cut.

The problem of designing functions of more than a simple logic block is another problem. System manufacturers were faced with this problem in attempting to find a minimum type-count p.c. board set. These nearly always ended up being a card full of distinct logic functions with a very small amount of logic interconnection on the card. This problem is being attacked even harder now since low-cost conventional IC's can create severe pin limiting problems on small p.c. boards. Connector and back-plane wiring costs are getting pretty tough to take.

High complexity IC's are forcing the semiconductor manufacturer into the same dilemma. He is finding room on the chip for lots of hardware but has to interconnect it to meet pin restrictions. The problem of the system man wanting a large variety of different interconnected functions for system optimization and the semiconductor man wanting to

produce a minimum number of different things for production efficiency is the problem that must be faced.

ARRAY ENGINEERING APPROACHES

In order to appreciate the system/semiconductor interface problem it is useful to review some approaches presently being taken in complex IC's. Three distinguishable approaches seem to exist, each with advantages and disadvantages.

Specialized Design

The most obvious method of design for a complex IC is to design a circuit to perform the desired function. Figure 1 illustrates a decade counter chip and wafer designed to perform a specific function. In this approach the circuit and each device in the circuit is optimized to be the simplest possible to meet the desired terminal characteristics.

This approach is great fun (and guaranteed employment) for the integrated circuit designers but is a process man's nightmare. It is expensive to design and to learn to make but it ultimately is inexpensive to produce. It's the best way to go on a very high-volume product. It's easier to take this approach on MOS circuitry since MOS logic circuits are simpler than bipolar logic circuits. This approach will be used widely on standard products and on custom devices where volume warrants.

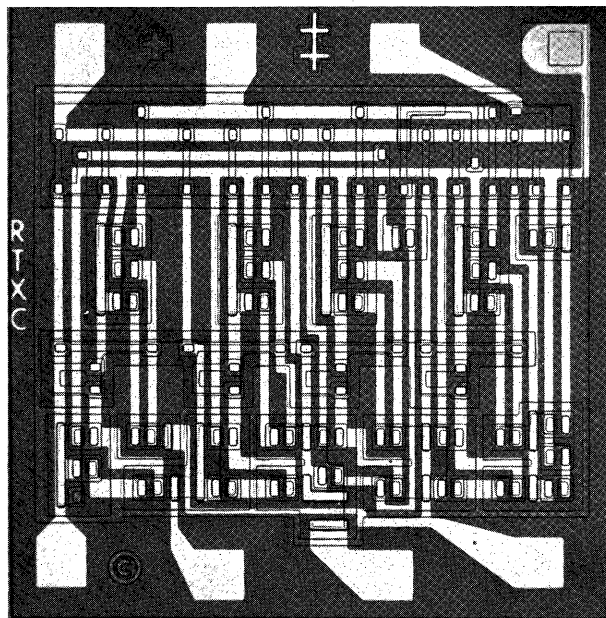


Figure 1. Decade counter.

Test and Connect

The test and connect approach to complex IC's was cued by looking at a wafer like Fig. 2 where the black dots indicate bad devices determined by testing on the wafer. Why go to all the effort of cutting the wafer into pieces, packaging the dice, and putting them on an interconnecting p.c. board when multilayer metal can connect them on the wafer? This is a valid question and the interconnection can be performed. This approach can provide very high complexity with modest technology but it has some disadvantages. First of all, pad area must be provided to perform the testing. Pads can typically amount to half the area of a present day conventional IC and would be most of the area of a small geometry IC. Second, at least four processing steps must be performed after testing which must be 100% good to interconnect the function. Third, the yield pattern is statistical therefore, the same function may have as many interconnect patterns as devices fabricated.

The test and connect approach and redundancy in general can provide monolithic IC's of higher complexity than any other approach but is not necessarily the most economical approach if the equivalent function can be fabricated otherwise. One significant feature of this approach is that the interior gates of the complex IC can be thoroughly evaluated, a feature that is difficult to accomplish and reduces the advantages of other approaches.

With respect to the IC user this approach has some interesting features. Design is at the simple logic block level since each device is like a conventional IC but there is probably only one kind of element available. A mix of gate devices and flip-flop devices on the wafer is possible but life gets much more complicated in requiring the good devices to match the desired mix. Custom interconnections of the gates to make custom functions is a minor perturbation to a system where two identical devices may have different interconnection patterns anyway.

Cellular

The cellular approach is illustrated by the photograph of Fig. 3. Each cell has a fixed set of components but the components may be connected into one of several possible logic block circuits by cell intraconnection. In this case, the cell may be a dual rank flip-flop, a quad gate, etc. The cells are interconnected on two layers. One layer of interconnec-

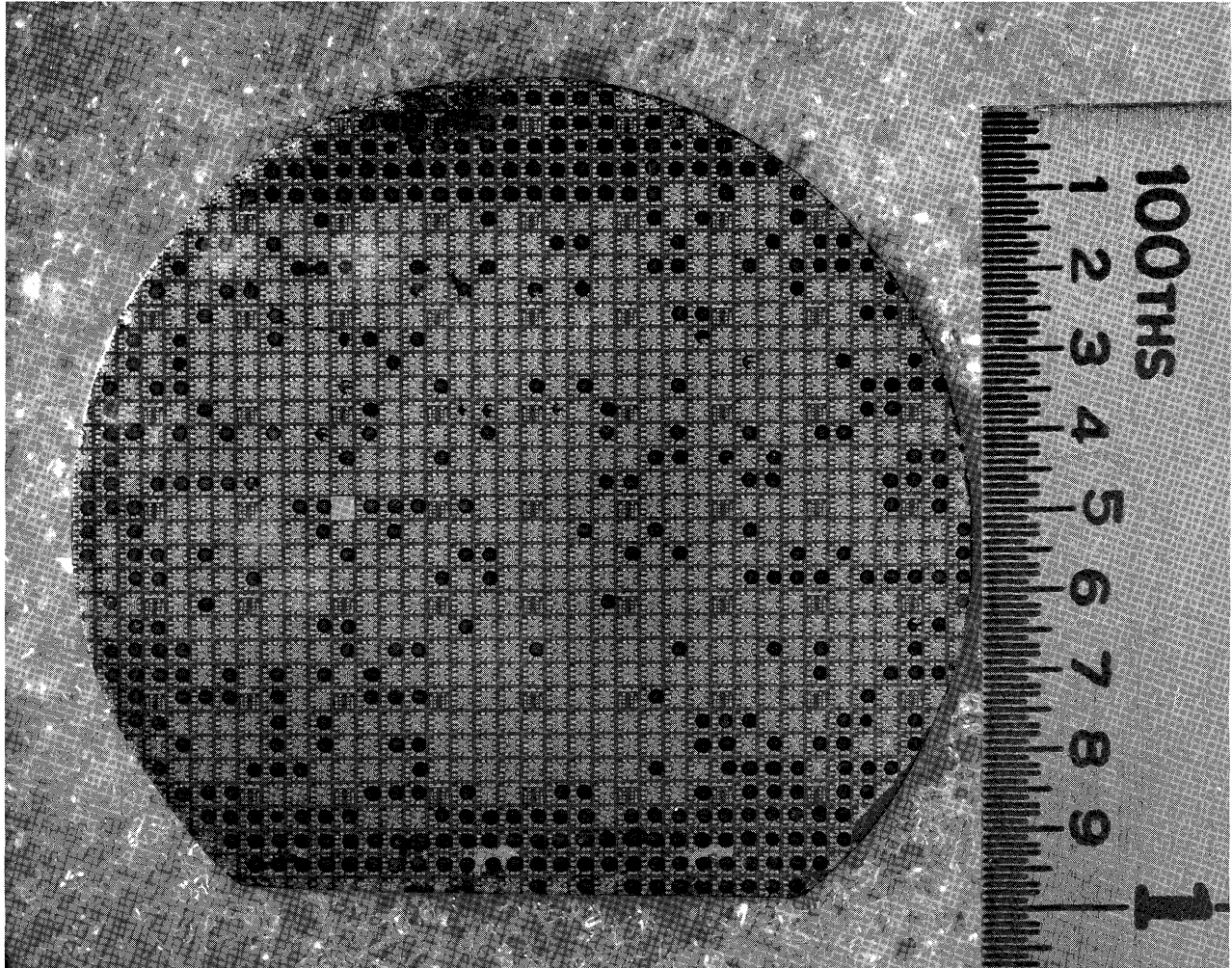


Figure 2. Mapped wafer.

tion is shared with the cell intraconnection, thus requiring a total of only two layers of metal. The approach is analogous to a p.c. board for conventional IC's where any member of an IC circuit family may be placed in any of a fixed set of locations and interconnected by the p.c. board.

This approach is distinguished from the test and connect approach in that no testing is done at the cell level, a selection of cell types is available and the interconnection pattern is fixed for a given function.

The interconnection patterns for this device were generated by taping exactly as would be done for a p.c. board. Wafers can be stockpiled presenting the processing man a much easier task since everything looks alike.

The significant characteristics to the user are that he can design at the simple logic block level exactly

as he does with a conventional IC family. Since customization is at the interconnect level with partly processed stockpiling permitted, the "tooling" cost and turn-around time for custom functions is reasonable. Similarly, the semiconductor manufacturer can use this approach to generate standard functions and sample them to customers without a large investment. In both cases a special design can be generated later if volume warrants.

THE SYSTEM/SEMICONDUCTOR INTERFACE

Within the framework of the technology described above, several possible divisions of effort and responsibility are possible. A few of the interfaces and the problems associated with them are discussed now.

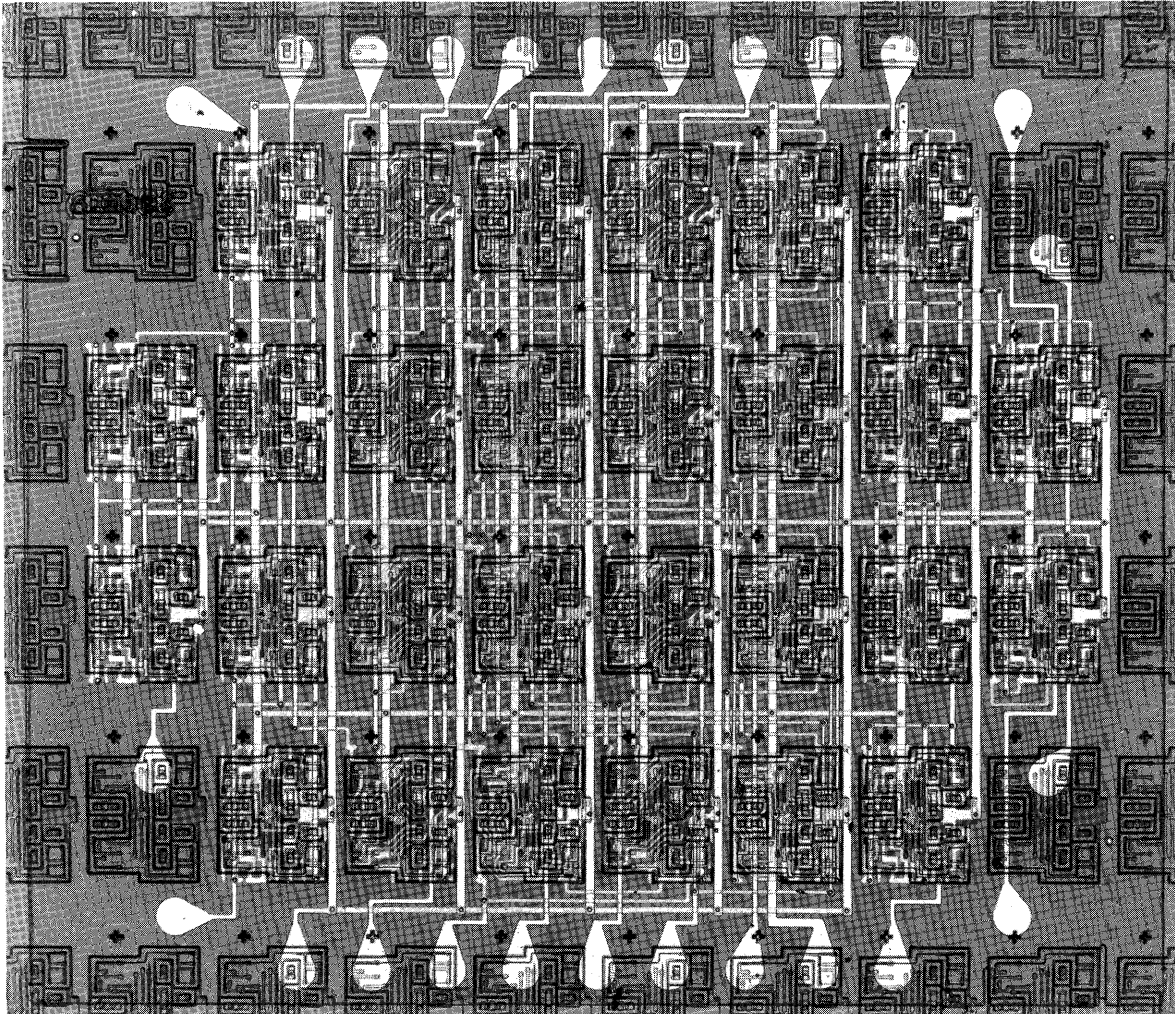


Figure 3. Cellular array.

Standard Products

The simplest interface (Fig. 4) is the use of standard products designed and fabricated by the semiconductor manufacturer. It has the advantages of fast delivery and low per package cost. It's the kind of situation that the semiconductor industry is best geared for today.

The disadvantages are most severe to the large computer manufacturer who loses control of performance and who has enough volume that standard products may not be the lowest-cost solution.

Testing and quality assurance problems can be worked out and spread over a large base of production.

Black Box Specification Interface

An interface at the black box level (Fig. 5) is a situation with the semiconductor manufacturer act-

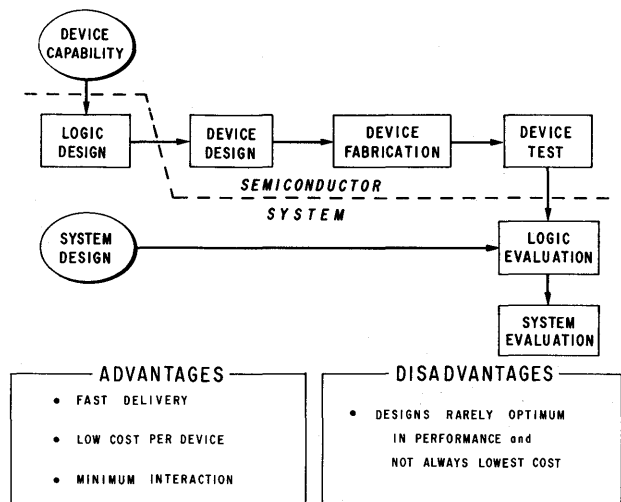


Figure 4. Interface using standard products.

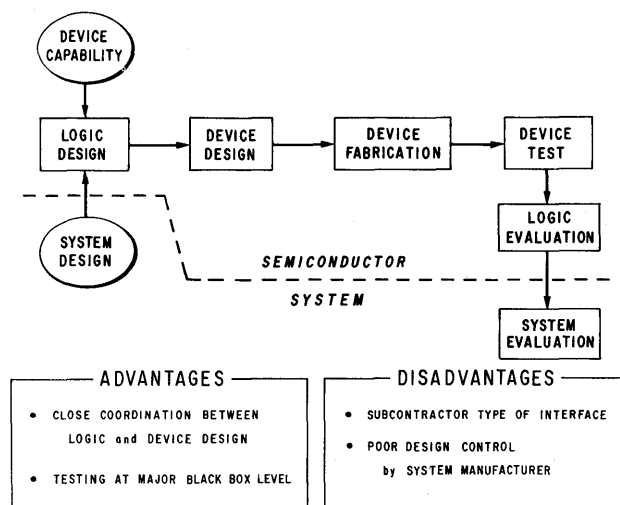


Figure 5. Interface at subsystem specification.

ing more as a subcontractor than a component supplier. The system designer defines a black box specification for the semiconductor designer to follow. Logic and device design can be quite well coordinated but the semiconductor manufacturer must build up a very sophisticated design staff to cope with such work.

In this situation testing can be performed at a major subsystem level thus significantly easing that problem.

Logic Block Diagram

One of the most intriguing interfaces is the logic block diagram level (Fig. 6). This interface is most apparent with the cellular or test and interconnect structures described above. The cell characteristics are defined much as they are today with conventional IC families. The system logic designers use this data to generate the logic block diagrams or equations describing the required functions exactly as is presently done. The logic description is then used to define the chip interconnections. The loss of control of interconnections on the chip is not critical since the dimensions involved eliminate many of the usual reasons for desiring control of interconnections. With a large volume of custom logic designs the semiconductor manufacturer can afford design automation for the mask-making.

The primary problem with this interface is the design of tests. The test sequence required for the complex IC's must be spelled out to the satisfaction

of both parties. This could easily be a more difficult job than the design of the masks.

Mask Design

The final interface discussed here is for the system manufacturer to design the masks (Fig. 7) either in the form of a meaningful drawing or the physical artwork ready for reduction. This puts the system manufacturer in complete control of the situation but makes it very difficult for the semiconductor manufacturer to factor in process improvements or control.

Mask design can apply at the cell interconnect level or at the circuit level although it seems more reasonable to stay at the cell level except possibly for MOS technology.

The testing problem is very interesting because the system manufacturer is the only one in a position to generate the tests but the semiconductor manufacturer would like to verify that the test and design are compatible.

CONCLUSIONS

The most universal interface problem is testing. Complete test of the function as a sequential machine is impractical, in general. The device must be tested against possible failures^{1,2} and with complex IC's, shorts and opens in the interconnection pattern are potential failures so that an intimate knowledge of the interconnect pattern is required to design the test.

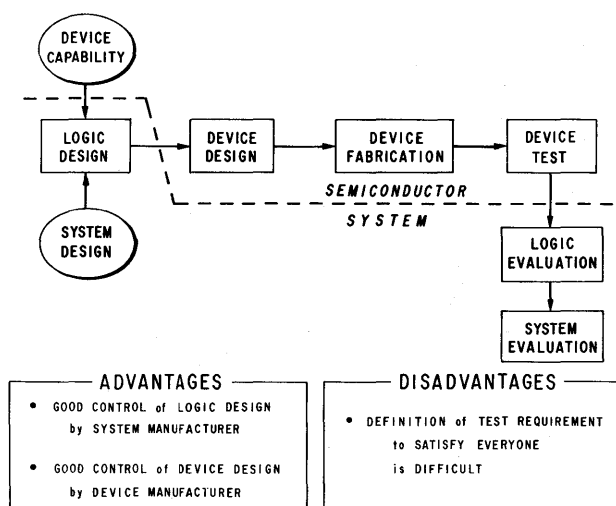


Figure 6. Interface at logic description.

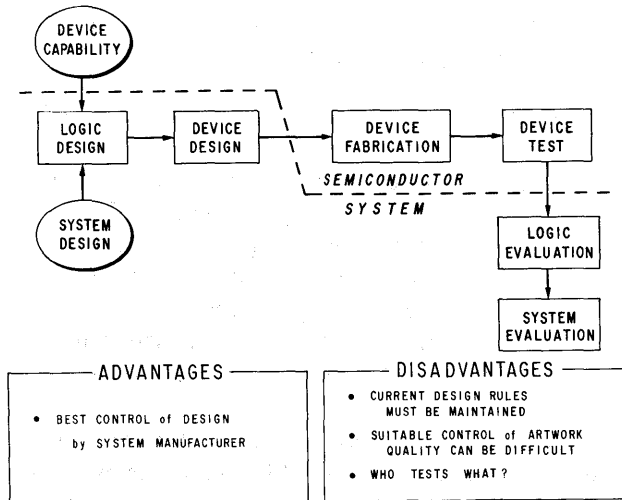


Figure 7. Interface at mask design.

An interesting facet of the interface is that a large volume of custom designs by the semiconductor manufacturer will justify use of computer-aids in logic design, mask-making and test sequence genera-

tion. This can effectively act as a service to the small system manufacturer who does not have the design volume to justify this kind of investment.

The selection of the best interface will depend largely on the job to be done and the capabilities of the user. For jobs with low volume the standard product is the best approach. If the user has limited logic design capability, the black box level would be interesting. The large computer users will be more interested in the logic block and mask design levels.

REFERENCES

1. H. Y. Chang, "An Algorithm for Selecting an Optimum Set of Diagnostic Tests," *IEEE Trans. on Electronic Computers*, vol. EC-14, no. 5 (Oct. 1965).
2. D. B. Armstrong, "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets," *IEEE Trans. on Electronic Computers*, vol. EC-15, no. 1 (Feb. 1966).

A LOOK AT FUTURE COSTS OF LARGE INTEGRATED ARRAYS

Robert N. Noyce

*Fairchild Camera & Instrument Corporation
Mountain View, California*

INTRODUCTION

Technology in integrated circuits has advanced to the point where we are considering integration on a substantially larger scale than is done today. In a companion paper to this one, R. L. Petritz has discussed the technological basis underlying large scale integration. It is the purpose of this paper to briefly review the motivation for the electronics industry to proceed to higher level integration, and to discuss in particular the cost implications of large scale integration.

Change in the method of producing electronic equipment will not occur without this motivation, this driving force. In the past, motivation has been provided by: 1) improved performance; 2) improved reliability; or 3) improved costs. If total costs are considered, the third point probably covers the first two, although the proper assignments of costs are difficult to make. Petritz has pointed out the advantages in performance and reliability which may be expected as a result of large scale integration. Let us here attempt to look at the last motivation—*cost*.

HISTORICAL COSTS

Actual basic production costs have been, for the most part, the principal determinant of total costs

in the components industry. A component was designed, and then manufactured in such extremely large numbers that the amortization of design costs was not a major factor in the total cost. Using the semiconductor industry as an example of one of the more sophisticated components industries, where a relatively large investment in research, development, and production engineering is required, the amortization of these investments is only about 10% of the total cost. When an increase in these investments resulted in a small percentage decrease in the basic production costs, it netted an overall cost reduction. It was the probability of this cost reduction that spurred the changeover from discrete transistors to integrated circuits.

A great deal of research, therefore, was invested to bring the technology to the point that practical yields could be achieved in integrated circuits, where the total labor, material, and manufacturing overhead was substantially reduced on a per device basis. As you know, this was done by eliminating the necessity of separating the individual components on the diffused wafer and individually testing and assembling them. Since the cost of processing an individual wafer through the diffusion process does not vary a great deal as a function of what is on that wafer, the basic production cost decreases as more can be packed on the wafer.

Now the industry is considering going a step further in integration, from integrated circuits to integrated arrays. Here again, production costs can be significantly reduced on a per circuit basis only by first achieving reasonable yields, then getting more circuits per wafer, and then reducing the total assembly costs. Assuming reasonable yields can be achieved, basic production costs could be reduced by virtue of more sophisticated design, probably by a factor of ten. Thus, there is enormous economic motivation to find a method of realizing these savings.

At the same time, however, another factor has been pushing up total costs: the start-up costs of producing the integrated circuits needed by the industry. These costs are becoming a significant proportion of the total cost of supplying integrated circuits to the user.

In particular, for integrated circuits designed to specific customer requirements, the start-up costs associated with making a custom circuit amount to approximately 30% of the total costs. This is an historical number from recent experience, but extrapolation of this experience leads us to conclude that integration on a larger scale is impossible until methods of reducing these critical custom start-up costs are found, even if we assume that the basic production costs are reduced to a negligibly small number.

CLASSIFICATION OF TOTAL COSTS

For the purpose of illustrating this point, let us divide the operating costs of a facility producing large arrays of integrated circuits into two classifications:

- a. Basic production costs which are incurred only once, the benefit being felt over all products.
- b. Custom costs which are incurred again each time a distinct product is made.

This classification is not, of course, a clear dichotomy, but for simplification we shall assume that it is.

- a. In the category of costs which benefit all products, we might include:
 1. Cost of total facilities and equipment.
 2. Process development costs.
 3. Technology development costs.

4. Costs of efforts aimed at reducing costs in Category b, such as design automation.
- b. The costs incurred for each product might include:
 1. Design of the array.
 2. Determining the specification necessary for a useful product.
 3. Devising a test program to assure that the specification is met.
 4. Tooling for this product, such as mask making.
 5. Certain inventory costs.
 6. Certain overhead costs.

Passing over those basic production costs in Category a, let us consider what happens to costs in Category b as we go to large arrays. If we assume that these custom costs remain constant per array, even though the particular array of circuits being produced is many times more complex than the integrated circuit of today, these costs will rise in direct proportion to the number of different arrays being introduced. As mentioned above, in recent history, start-up costs of custom designed integrated circuits have amounted to 30% of the total; thus, if we must supply only somewhat over three times as many different arrays as we now supply circuits, these custom costs alone will be more than our total costs of supplying the customer need today.

I have used in this example the business of supplying circuits to specific customer design, which you may consider unfair. In this case, we may look at the standard microcircuits as they exist today, where Category b costs are a lower percentage since production volume is higher. In this case, these costs amount to about 10% of the total costs. Thus, if the number of different arrays which must be furnished increases by a factor of ten, then again these custom costs alone will be equal to our total costs of supplying the customer today.

Obviously, we have trouble finding economic motivation for the use of large arrays if we greatly increase the number of arrays used and continue to operate in the same manner that we do today.

HOW MANY DIFFERENT ARRAYS?

It is well known that a logic system can be made up entirely of simple identical gates, and we have examples of computers where this has been done, such as the Apollo guidance computer. However, in

digital integrated circuits, we have found it advantageous to go further than this and to include various different gates and flip-flops, for example. Most systems being built today utilize from three to ten different integrated circuits. As the level of integration increases, we expect that this number of distinctly different individual array configurations will increase drastically.

Perhaps the closest example we have today of a system being built using large arrays is the IHAAS computer using MEMA (MicroElectronic Module Assembly) modules. In this system, six different integrated circuits of today's complexity are used. These are assembled into MEMA's, each consisting of about twenty-five integrated circuits, and then encapsulated. Over sixty distinctly different MEMA's are used in the system. Thus, the number of different components used in the system has increased by a factor of ten in going from the integrated circuit to the array of integrated circuits. If higher levels of integration were sought, this ratio would probably increase until the level of an entire subsystem were reached. On the basis of this example alone, we would expect that we would be producing ten times as many different arrays as we are now producing integrated circuits. Other systems which might use the same six integrated circuits at present levels of integration could be expected to add different arrays, resulting in far more than ten times as many configurations.

CUSTOM ARRAYS OR STANDARD ARRAYS?

Even with the significantly higher number of arrays needed and the present custom start-up costs, we might still have a chance of bettering today's costs if identical products could be furnished to large segments of the industry. I believe that the reaction to this suggestion will be similar to the reaction that greeted the suggestion that standardized integrated circuits be used.

It has been demonstrated, of course, that identical integrated circuits can be used widely in the industry. This has allowed the potential cost savings of integrated circuits to be realized by the industry. It is clear that the economic motivation for such standardization is greater now than it has been in the past and, therefore, the industry will surely move in this direction. How far the industry moves, however, may very well determine whether or not large arrays are used in significant quantity in the future. Some

possible standard arrays can be seen today, particularly memories and shift registers. The appearance of more standard arrays seems inevitable.

PLAYING WITH NUMBERS

The component supplier has traditionally been asked to test his product extensively to assure that it meets a specification that guarantees its usefulness to the buyer. This will be possible only with some limitations in regard to large arrays.

Let us assume that we have a ten-by-ten array of elements, each of which may assume either of two states. To test straightforwardly every combination in this array would require 2^{100} different tests. Even if testing rates could be increased to 10^8 tests per second, a complete functional test would require 10^{16} years! There are ways around this, of course, by breaking the array into smaller units to test. In this example, if ten groups of ten elements each were tested, an exhaustive test would require 10×2^{10} tests, or 10,240 tests. This is possible. To do such testing would require that access be provided to intermediate points of the array, which is again quite possible. Functional tests of this magnitude are practical with equipment now available. However, exhaustive parameter tests of this magnitude would be prohibitively time-consuming.

The difficulties of exhaustive testing will also, in all probability, initiate a change in the way that specifications for large arrays are worked out between customer and supplier. Instead of being outlined in a written document, as in the past, specifications will be built into a complete test program. Consideration of how the large integrated array will be tested will be an important part of its design.

WHAT LEVEL OF COSTS IS NECESSARY?

An implied assumption has been made throughout this discussion that the *component* costs using large arrays must be less than the *component* costs using integrated circuits before integrated arrays become economic to use. This is not strictly correct, since some additional cost reductions will accrue to the systems manufacturer using large arrays. However, if the changeover from discrete transistors to integrated circuits serves as a good example, the actual component costs *will* have to be nearly comparable before widespread use occurs.

The motivations other than cost, i.e., performance, reliability, size, and weight, will promote their use in special situations before the required cost level is met.

HOW DO WE PROCEED?

The primary motivation for proceeding to develop large scale arrays has been the promise of reducing the labor needed to produce that function. But we have seen that the straightforward extension of the methods for designing and producing integrated circuits is economically impossible unless some standardization can be achieved. If not, the custom start-up costs for each array will be more than the total cost to produce it by other techniques. We come to this conclusion even presuming that basic production costs are negligibly small, and that the design costs for the integrated arrays will be no greater than for the much simpler integrated circuits of today.

Still, the motivation to find a way around this problem is great, because of the potential savings in basic manufacturing costs. So let us look again at the origin of these custom costs.

Here we can take the solution which the systems designers have been using for years, and apply it to the component business. We have argued that standardization must be achieved if arrays are to be made economically, and have tacitly assumed that this standardization must be at the complete array level. Attempts to do this in system design have been largely unsuccessful in the past. We have seen standardization at the level of the basic gate, however, and this is certainly the clue to the solution of our problem.

As a matter of fact, if we examine closely where the custom costs for a new integrated circuit family arise, we find that the cost of the basic gate is by far the highest, with the other elements following relatively inexpensively. Can we not standardize on the basic element, and let the rest of the integrated array be designed to the particular requirement?

There is, of course, a great deal of design work beyond this level, but here the component manufacturers can draw on much that has been accomplished by the equipment manufacturers. The design of masks for interconnecting basic gates is not much different than the problem of laying out printed circuit boards, even though the constraints may be different. We expect that we will have to rely on the computer to do this job, just as the computer manufacturers themselves have done. In this manner, we trade costs in Category b for costs in Category a. The complete design automation software will have to go much further. Starting with logic reduction, after determining what elements are necessary, it must give us a graphic output of the interconnection patterns, and devise a test program for assuring that the final product is adequate. These programs are well underway, and are scheduled to provide positive results by the time the processing technology for making large integrated arrays is really available.

CONCLUSIONS

Basic production costs, such as yield improvement, will be substantial for a family of large arrays. However, they will be incurred only once, and can be amortized over the entire product line, so they will not be so critical.

Custom design and test costs, on the other hand, are going to be very critical. These costs, occurring each time a new array is produced, will preclude widespread use of arrays until they can be drastically reduced. Standardization will help to reduce these custom start-up costs, either at the array level or, more likely, at the level of the basic gate. Also, automatic techniques for array layout, tooling, and testing must be perfected before arrays can be produced economically in quantity.

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.

A MULTIPROGRAMMED TELEPROCESSING SYSTEM FOR COMPUTER TYPESETTING

B. E. Nebel

Los Angeles Times
Los Angeles, California

INTRODUCTION

The use of computers to hyphenate and justify printed material is widely accepted in the printing and publishing industry, as well as in institutions conducting research in literary analysis and photo-composition.¹⁻⁴ Newspapers are among those using computers in the production of type cast in lead. The staff of the *Los Angeles Times*, in collaboration with RCA Corporation, implemented a program for the RCA 301 to accept typewriter by-product paper tape at 1,000 characters per second, to justify and hyphenate lineage of variable font size to newspaper column width, and to produce paper tape output used to drive automatic hot lead linecasters. This set of programs has been supporting typesetting production requirements of over 300,000 lines per week for the past three years. Additional requirements, both present and future, demanded that a more universal use of the computer as a system with teleprocessing capability be implemented. The three principles underlying these requirements were:

1. Immediate recovery capability for operational failure and hardware demise.
2. Centralization of the computational function to allow remote news sources and printing facilities the use of the automatic typesetting resources.

3. Potential to expand the system allowing on-line graphic display terminals and time-sharing with large business applications.

Twin IBM System 360/Model 30's were chosen as base processors with a suitable peripheral configuration to satisfy at least the first two requirements. Expansion was considered a reasonable gamble when accepting the alleged upward compatibility of the system. The necessity of multiprogramming capability to service even slow-speed paper tape terminals, as well as the unavailability of software to support either these devices or a multiprogrammed mode of operation, was recognized and became the major in-house systems programming project.

The purpose of this paper is to describe that portion of the typesetting system dealing with paper tape terminal service and multiprogramming implementation, with the hope that it will demonstrate the feasibility of both teleprocessing and multiprogramming techniques for a typesetting application on a small computer within a basic operating system configuration. There is no attempt here to examine that portion dealing with type composition. The complexities of the typesetting techniques could easily be the subject of another paper and are certainly beyond the scope of this presentation.

SYSTEMS PROGRAMMING OBJECTIVES

The objectives set forth during the planning stages of the project are:

1. *Support existing typesetting hot metal production requirements.*
2. *Implement a multiprogramming system to:*
 - Accommodate on-line photo-composition programs.
 - Facilitate additions and changes of application programs.
 - Permit multi-task processing.
 - Gain maximum speed from asynchronous input devices.
3. *Provide telecommunication capability for:*
 - Remote Printing requirements.
 - Economic interchangeability of typesetting terminal devices.
4. *Adopt methods compatible for transition to higher level systems.*

These may be summarized as:

- Implement telecommunication system capable of expansion with growing application areas and yet of simple enough construction to require minimum additional standards and rules to be imposed upon the applications programmer.
- Implement a multiprogramming system utilizing as much of the distributed software as possible; making minimum modifications; and adhering to a philosophy compatible with larger systems.
- Implement file design and job flow techniques which will allow load balancing between the twin processors when a channel-to-channel adaptor level is reached.

When posed with a Model 30, 32K, 2 μ sec per byte memory hardware level, and Disk 8K BOS supervisory software level, these objectives, coupled with the typesetting requirements, were at once ambitious enough to arouse great interest among the staff and restrictive enough in scope to permit a reasonable design pace.*

As a result of the stated objectives, 8K BOS constraints, and the choice of configuration, it was decided at the outset to restrict the supervisor

* Core storage was later expanded to 64K, 1.5 μ sec per byte memory.

modification to (1) augmenting the multiplexor channel scheduling to permit device (terminal) scheduling, (2) intercepting selector channel I/ \emptyset interrupts only when pertaining to a single logical unit data set so as to allow relative freedom of the processor programs to use the existing logical I \emptyset CS for private data set definition.

Regarding the multiprogramming requirement, it was decided to adopt the more primitive commutator program scheduling techniques on the grounds that they are easier to implement than the more advanced C.P.U. list methods, are more in keeping with the pre-load bound variable technique used by the 8K BOS Linkage Editor, and can be adapted[†] from existing systems.^{5,6}

SYSTEM CONFIGURATION

Hardware

The hardware configuration for the present system implemented at the *Los Angeles Times*, as illustrated in Fig. 1, is described below.

The "front end" teleprocessing equipment is composed of TTS, IBM and DuraMach typewriter-perforators operated off-line by typists trained in the preparation of textual material properly coded for the hyphenation-justification programs. The prepared tape (with typist identification) is transferred by an operator to Teletype CX Paper Tape Readers attached to IBM 2972 Line Control Units. The 2972 Control Unit appears to the C.P.U. as an IBM 1030 type device and is polled across full duplex telephone common carriers terminated at each end by Western Electric 202D Data Set modems. Each 2972 Control Unit allows up to 18 attached devices, but at present with half duplex mode only two devices (paper tape reader and punch) are attached to each of seven control units.

Interfacing with the Model 30 C.P.U. are two IBM 2702 Transmission Control Adaptors capable of transmission speeds of up to 60 cps on from 1 to 16 lines simultaneously with 24 unique addresses per line. The 2702 is equipped with a two-channel program controlled switch making the adaptors accessible from either of the two C.P.U.'s. In the Model 30 configuration, the channel switch feature is used primarily for backup and recovery purposes in the event of C.P.U. failure. The twin Model 30's

[†] The IBM 7040/7090 Direct Couple System was the principal basis for the adaptation.

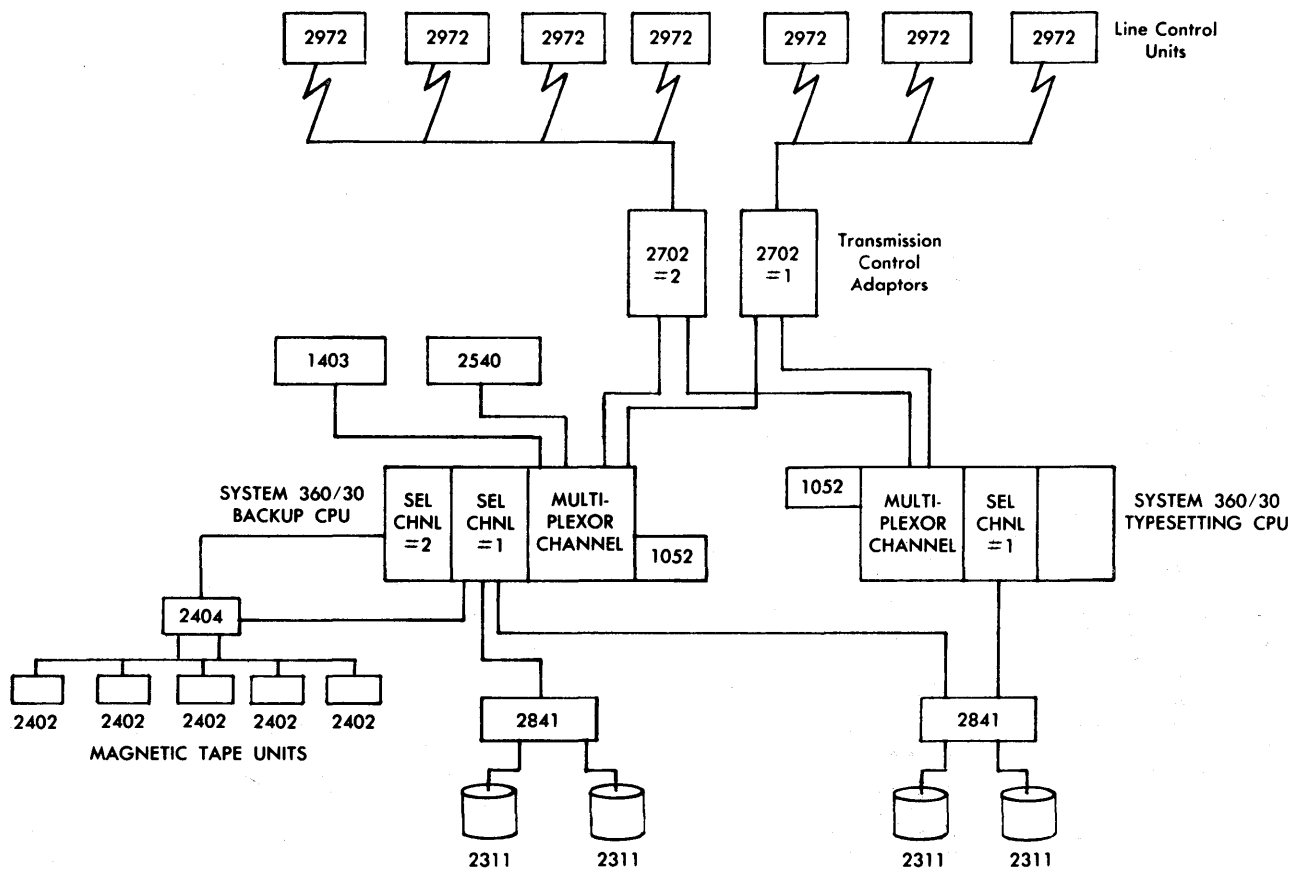


Figure 1. Typesetting hardware configuration.

are identical except that the backup C.P.U. has two selector channels and the 1401 compatibility feature. The backup C.P.U. normally performs business data processing installation functions with a peripheral configuration, including: a 2404 read-while-write tape control unit with five 2402 9-track magnetic tape drives; a separate 2841 control unit with three attached 2311 disk drives; a 1403 printer; and a 2540-card read/punch. The typesetting C.P.U. has as local backup storage two 2311 disk drives under control of a 2841 Control Unit (on one selector channel) again with a program controlled switch making it accessible from either C.P.U.

This configuration, both from the standpoint of typesetting and commercial applications, is judged economically sound and a reasonable starter system configuration for the design objectives.

Software

Referring to Table 1, the software configuration may be described in terms of the program control and typesetting system functions as:

1. 8K BOS (Disk) supervisor modified to include 2702/2972 terminal service with device scheduling.
2. Typesetting system control program including multiprogram scheduling, multi-task management, disk I/O control, direct access methods with dynamic track allocation, data buffer pool management, and system initialization or recovery processes.
3. Telecommunication service programs providing message initialization, identification, blocking and recovery processing.
4. Communication editing programs providing message translation, routing and task initialization.
5. Typesetting application programs including all hyphenation and justification capability, wire service processing, and production accounting processing.

Table 1. Program Control and Typesetting System Functions

8K Basic Operating System Supervisor	Job Control System Loader Channel Scheduler Physical IOCS Interrupt Processing
Supervisor Expansion for Teleprocessing	Multiplexor Device Scheduling Terminal I/O Control Input Device Polling Terminal Error Routines
Commutator	Multiprogram Scheduling Save and Restore Functions Overlay Control
System Control Resident Programs Task Control Table and Master Track Allocator	Multitask Scheduling Global Data Set I/O Queuing Dynamic Track Allocation Task Purging System Initialization System Recovery
Global Data Set I/O Processor	Global Data Set I/O Scheduling Interrupt Synchronization Priority I/O Functions
Checkpoint Processor	System Check Point Scheduling
Teleprocessing Terminal Service Processor	Terminal Control Block Maintenance Line Turnaround Scheduling Interrupt Synchronization Error Recovery Routines
Communication Editing Processor	Message Translation Task Routing Message Error Detection Task Initialization
Overlay Area Hyphenation/Justification Processors Library Maintenance and Retrieval Processors	Typesetting Processing Message Splitting Production Accounting Library Retrieval and Take Correction Functions
Buffer Pool	

- Library maintenance providing the transfer of processed "takes" to magnetic tape backup files for historical and retrieval purposes.

TAILORED CONTROL PROGRAM CONSIDERATIONS

Commutator

The commutator is assembled from a set of GATE macro instructions, each expanding into a routine providing uniquely named program exit points for system sharing and program deactivation. Register saving and restoring, and program entry point linkages, are provided automatically when required. The macro parameter OVERLAY and AREA cause the generation of coding to fetch the

program phase specified by the macro parameter NAME. Program execution is scheduled by constant polling of the commutator gates where the physical position in the list determines the program priority. If a gate is "open" control is passed to the named program, otherwise the polling continues sequentially. An accelerator switch is provided to change the course of the polling in case a high priority program requires immediate control. The activity status of the entire commutator is examined at the end of each polling operation to prevent the system from clocking time when all functions are complete. Under these conditions the wait state is entered until an external interrupt causes polling to begin again.

Program Sharing

It is incumbent upon the systems and applications programmer to share control of the system when waiting for some asynchronous event to occur or be completed. The programmer releases control to the system during a waiting period by issuing a SHARE macro instruction which generates the necessary linkage to the share exit point in his commutator gate routine. The housekeeping necessary for proper reentrance is performed by the commutator routine. When the task segment is complete, the programmer deactivates his program by issuing a DONE macro instruction which simply provides linkage to that exit point in the commutator routine which closes the appropriate gate.

Data Buffering

A common (global) buffer pool provides fixed length buffers arranged as a simple push-down list. At present the global data set is the only one available to the programmer and therefore file design must be made to conform to it. Efforts are now under way to provide the programmer with the ability to specify private (local) pools of variable length buffers allowing more freedom in handling pre-prepared volumes without setup requirements.

Task Priority

The system tasks are introduced at the 2972 terminals in the form of paper tape messages. Each message, a priori, constitutes a task which is added to the Task Control Table by the terminal service programs. Task Control Table entries are ordered on a first-in-first-out (FIFO) basis. Since the type-

setting applications generally do not require express action for one segment of work over another, there is presently no priority of tasks established in the Task Control Table structure. Each task is staged appropriately for the processor program which it requires. Consequently the stage mix at any time represents, and can be measured as, the contention factors for the system resources. The programmer acquires tasks by the use of the GETSK macro in which he specifies his own stage as a parameter. When a segment of work has been completed on a task, the programmer stages the task for another processor by use of the PUTSK macro and the OPNGT macro which opens the appropriate commutator gate.

Track Allocation

The global data set available to all programs is contained on one entire 2311 disk pack volume. The current status of all tracks on this volume is maintained in a master track allocator bit table, where each bit represents one track. An available track, when required, is obtained by subjecting the relative position of the available bit to a simple transformation algorithm which produces a cylinder/head address. Each Task Control Table entry contains track control bytes which provide the track identity. Tracks are allocated and returned to availability status dynamically under complete control of the programmer.

Overlay Areas

The typesetting system allows permanent overlay areas to be defined at Linkage Edit time. Application program phases specify into which area they are to be loaded by Linkage Edit phase parameters, and thereafter contend for C.P.U. control according to the commutator priority (the relative position of the application program's gate in the commutator).

8K BOS (DISK) MODIFICATIONS

The Basic Operating System was modified in some fairly minor areas, with the exception of major additions to the multiplexor channel scheduler. These are described below and depicted in Table 2.

Job Control was altered by the addition of a separate phase to allow the console option of calling for normal end of job (SYSEOJ) processing, the Type-

Table 2. IBM 8K BOS (Disk) Modifications

Job control	1052 Console job control options Specific program fetch control Program check console snapshot option
SYSDMP	Dump to disk
Multiplexor channel scheduler	Terminal control block Device scheduling Error recovery routines Polling of input devices
Selector channel scheduler	Global data set interrupts synchronized with system I/O
Additional SVC codes (supervisor call)	Terminal disabling by a system processor Separate start I/O capability for terminals

setting Control System, a special SYSDMP program, or any other phase from the core image library if specified by name. The last option is used primarily to fetch system and communication line diagnostic programs.

SYSDMP was modified to dump core to a scratch disk rather than to a printer. This was necessary since the typesetting C.P.U. does not have a printer attached to it when operating in the normal installation environment. The pack is switched to the back-up C.P.U. for off-line printing.

Multiplexor Channel Scheduler has undergone extensive modification to allow polling of telecommunication lines, to allow data chaining in the channel command word string facilitating asynchronous data transfer from each terminal for buffers of any length, to service the I/O interrupts from telecommunication terminals in coordination with the device scheduler and the buffer queuing requirements, and to selectively implement error recovery techniques on a real-time basis.

Selector Channel scheduling is unchanged except that I/O interrupts pertaining to the global system data set are intercepted in order to perform system I/O synchronization. All other selector channel activity is unmolested thereby giving the programmer freedom to define other *direct access* private data sets. The only restriction imposed for private data sets is that the WAITS typesetting system macro must be used instead of the logical IOCS WAITF macro. No program wait loops are permitted in the system.

Additional SVC Codes have been implemented to allow separate start I/O (SIO) capability for the

telecommunication terminals under device scheduling. SVC codes have also been added to permit the processor program to selectively stop any one or all paper tape readers in the midst of transmission. The latter codes are required if the system approaches saturation.

TELECOMMUNICATION SERVICE PROGRAMS

The telecommunication terminal service programs formulate and maintain Terminal Control Blocks for all enabled devices. The Terminal Control Block contains current status bytes indicating device availability, primary and secondary buffer pointers and buffer status bytes, error status on the line or device (sense IO bytes), and pointers to channel control word (CCW) strings associated with the device.

The telecommunication service programs maintain constant polling of all reader devices. When a reader starts transmitting data, the message is identified by date, line number and sequence number, and added as a task entry to the Task Control Table staged for the terminal editing routines. As soon as the primary buffer is filled, data is chained to the secondary buffer, the primary buffer is transmitted to the editing routines and the primary buffer becomes secondary; the secondary becomes primary.

When an end of message code terminates the input operation, the communication line is "turned around" and readied for output. When a task staged for punch output becomes available for the line, the service programs generate visual identification header information and subsequently punches the hyphenated/justified text enqueued from the disk file.

At termination of the punching operation the line is again "turned around" readied for input and polling is resumed.

SYSTEM RESIDENT FUNCTIONS

The system control resident programs perform task management, global data set disk I/O queuing, track allocation, task purging, system initialization, checkpoint and processor save and restore functions.

The meaning of a "task" to the communications editing programs differs from the meaning to other processors in that the editing program considers a full input buffer as a task whereas the other proces-

sors consider a task as an entire message. The editing routines are at any moment most likely processing messages from all terminals simultaneously. It is principally for this reason that "task local information" bytes were included in the Task Control Table entry as shown in Fig. 2. These bytes contain the track queuing information from buffer to buffer for each message. The other portions of the task entry are necessary to all processors. The date, originating device address, and "take" number are collectively termed the Task ID, carried throughout the processing and identified as such in the output punched visuals and the library directory for retrieval purposes. The "take" header track address points to the first record in the chain of queued records on the backup storage. The task stage number identifies the current stage of processing. The track control bytes specify the next available track for the task. The tasks are added (created), restaged, gotten and purged by the programmer by use of ADTSK, PUTSK, GETSK and PURGE macros. As tasks are finally staged for the library or purged, the entries are in the former case transferred to the library directory and then overlaid, or in the latter case simply overlaid, to make room for additional entries.

The global data set is used by the applications programmer when dealing with data which is either being passed to him from another processor or which he is creating for the future use of another processor. This data set is defined and opened by the system resident disk I/O processor. The programmer has access to the data set by use of the DSKRD and DSKWR macros and in special cases where a file update operation must be accelerated,

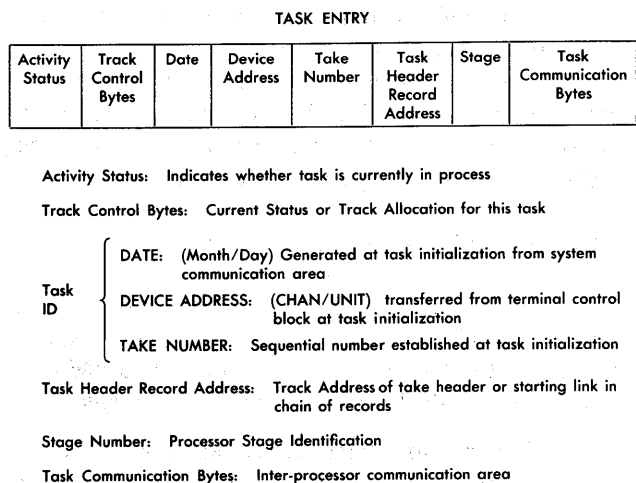


Figure 2. Task control table format.

by the priority write DSKPW macro. The DSKRD macro includes a parameter for posting the completion of the operation, thus serving as a "waiting" indicator during the course of program sharing. The control program maintains a separate queue for this data set, synchronizes the I/O operation and accelerates the execution by maintaining a high priority for control of the C.P.U.

Track allocation is accomplished through the use of the TRACK macro. The track control bytes in the task entry contain a pointer to a list of all tracks currently in use by that task. As tracks are allocated, they are made unavailable in the master track allocator and recorded in the track control list. When the task is subsequently purged, all tracks in the control list are returned to the available status in the master track allocator. The task entry itself is then overlayed and all trace removed from the system.

Whenever a task is added to the Task Control Table or the stage of a task is changed by means of the PUTSK macro (i.e., the last stage was successfully completed), a checkpoint is taken of the Task Control Table, the master track allocator, task control pointers and other information required for recovery of the system. The extent of this checkpoint, plus a minimal restart operational procedure has proved adequate in recovering system operation from the last checkpoint within five minutes from either a catastrophic C.P.U. failure which requires switching to the backup C.P.U. or an intermittent disk error which requires only a restart procedure at the terminals.

SYSTEM TASK FLOW SUMMARY

In summary, the course of a "take" through the system is described below in conjunction with the flow presented in Fig. 3.

The "take" information (message) immediately follows the 2972 device ready response to a polling operation. This peculiarity requires the acquisition of primary and secondary buffers prior to the polling of each terminal (line address). The physical message blocks (terminated by a carriage return code) are concatenated by the terminal service programs into a logical buffer record. A program controlled interrupt, caused by the first primary buffer of a message being filled, signals the terminal service programs to construct a task skeletal entry containing the task ID (date, device address, take number),

and add the entry to the Task Control Table staged for the Communication Editing programs. The editing programs, in turn, translate the message characters from the originating code to EBCDIC, examine the message parameters to determine future routing requirements, record this and any data error indications in the buffer control bytes, acquire a track (from the global data set) for the task, and queue the buffer record on disk. When the editing routines encounter an end-of-message code, an end-of-file condition is recorded at the end of the queued records and the task is staged for whichever application processor the edit programs decide is required. In most cases the takes require the action of the hyphenation/justification processors; however in some cases the messages may only be routed to a terminal other than the one from which it originated. The hyphenation/justification routines, when gaining C.P.U. control, process the input take and create (add) a task containing justified text in a form ready to drive the typesetting device. The output record chain is queued to disk again as part of the global data set, and the created task is staged for the terminal service output programs. The input task is normally staged for processing by the library maintenance programs but may in some cases be purged from the system at this point. The terminal service output programs, when gaining C.P.U. control, enqueue the output message, transmit it to the terminal specified by the routing information and, after disconnecting the terminal, purge the output task. Accounting information (line count, typist identification, error count, etc.) is recorded in the header record of the input task and processed during slack periods for accounting reports.

CONCLUSION

The adaptation of multiprogramming and teleprocessing techniques to a "small scale" computer system within the bounds of a distributed basic operating system was a prime objective in the *Los Angeles Times* Typesetting System project. The system has been operational only a short time and consequently no statistical system timing evaluations have been made. The system gross performance has so far led us to feel justified in attempting techniques of this kind even at a basic system level. It is our belief that the present typesetting requirements as well as those of the future will be met with versatility as a result of the high degree of programming modularity.

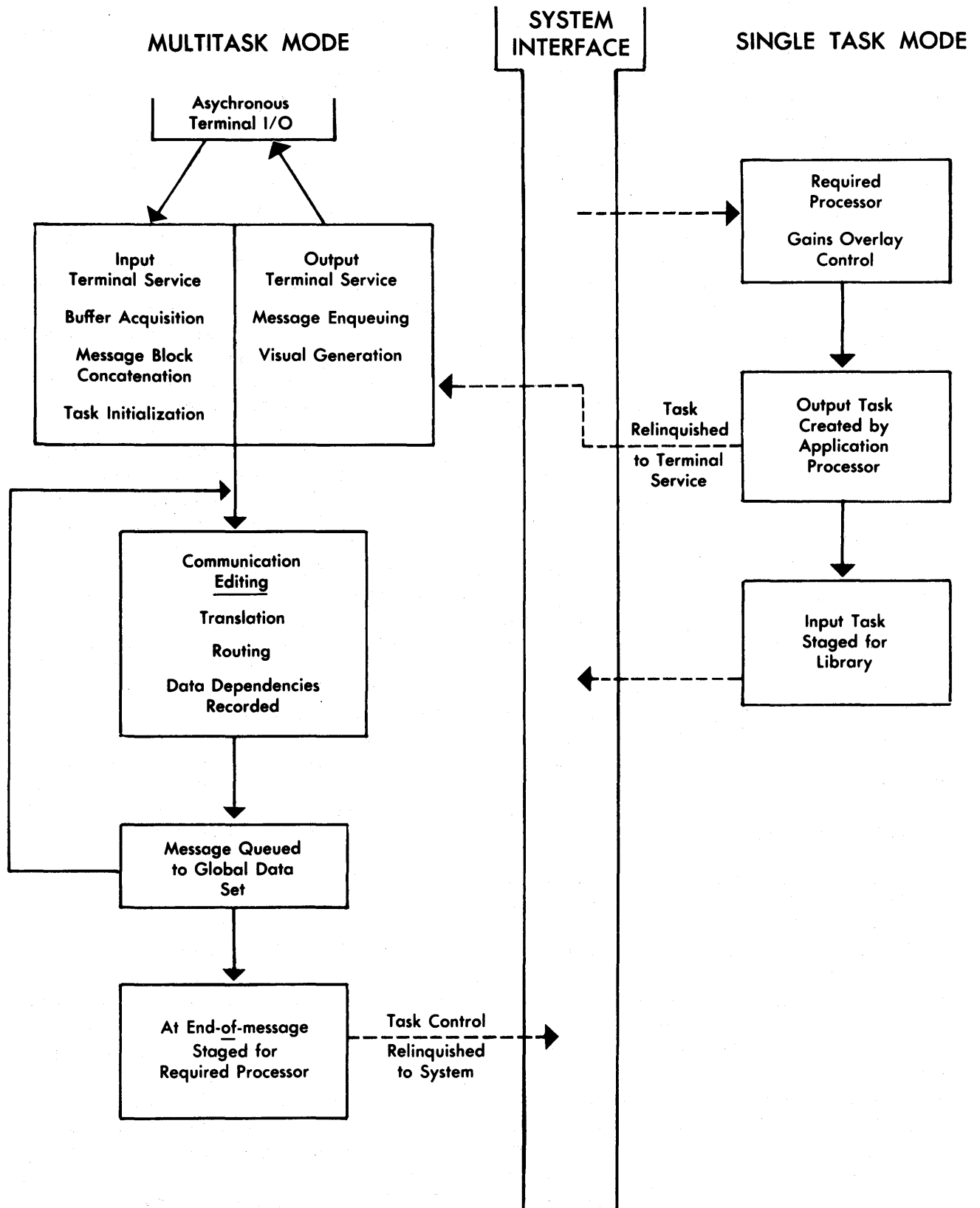


Figure 3. System task flow.

ACKNOWLEDGMENTS

The author wishes to acknowledge the work of the following persons, without whom the project could never have been accomplished: Lynn Abbott, *Los Angeles Times*, wrote the typesetting hyphenation programs; Joyce Stevens, *Los Angeles Times*, wrote the input message editing programs; Joseph Maloney, IBM, wrote the telecommunication terminal service programs and the multiplexor channel additions to 8K BOS; Gil Flores, IBM, wrote the typesetting justification programs.

REFERENCES

1. Lee Ohringer, "Progress in Computerized Typesetting," paper presented at the 17th Annual Meeting of the Technical Association of the Graphic Arts, June 1965.
2. M. V. Matthews and Joan E. Miller, "Computer Editing, Typesetting and Image Generation," *Proceedings 1965 FJCC*, vol. 27, Spartan Books, Washington, D.C., 1965.
3. P. F. Santarelli, "Computer Prepared Text: A Real-Time/Time-Sharing Multi-Terminal Publication System," Technical Report, IBM, SDD (Apr. 1965).
4. M. P. Barnett, *Computer Typesetting; Experiments and Prospects*, MIT Press, Cambridge, Mass., 1965.
5. R. V. Bergstresser, "Tutorial on the Attached Support Processor Multiprocessor Operating System," paper delivered to ASP Users Group at SHARE XXVI Convention, Mar. 1966.
6. William Desmond, *Real Time Data Processing Systems, Introductory Concepts*, Prentice-Hall, Englewood Cliffs, N.J., 1964.

INTEGRATED AUTOMATION IN NEWSPAPER AND BOOK PRODUCTION

John H. Perry, Jr.

*Perry Publications, Inc.
West Palm Beach, Florida*

For your background information—and so that you may better understand the scope of our automation effort—I should like to explain that our operations include the publishing of 27 newspapers and two magazines.

We also operate the statewide All-Florida News Service. We have offices in 35 cities and employ some 2,000 persons. Our commercial printing covers the full range from simple brochures to complicated catalogs and encyclopedias.

Making computer technology the very heartbeat of such an organization has been a gratifying experience. We feel that in moving toward the era of a total computer system, we have contributed leadership in the direction which an industry afflicted with ever-rising costs eventually must go.

Although our primary concern is with newspapers, unforeseen capabilities of the computer have enabled us to lease time for outside data processing and to embark on book production. So far, our computer equipment has been used to produce more than 3,000 books.

Perry Publications' use of electronic computers in printing and publishing began almost four years ago, when the Radio Corporation of America suggested we use a computer not only for accounting, as we had been doing for two years with IBM punched card equipment, but also for preparation of typesetting.

In those early days, there were few of us at Perry who had heard of such a thing as a nanosecond. But we have since learned.

At Perry, we use both the offset and letterpress processes; and both hot metal and cold type. Our early expectation was that automation would enable greater use of less-costly cold type. The expectation has come true.

To get us started, RCA set up a group of specialists to analyze our problems and work with our staff.

Our first computer was the RCA 301. Its primary purposes were bookkeeping, hyphenation and justification.

To obtain accurate hyphenation, we programmed the dictionary into the system's magnetic tape unit. However, this slowed the operation so much that the computer could be used only for straight-matter composition. To overcome this, we superimposed a logic system consisting of a series of phonetic tables which provided hyphenation.

We then decided that installation of a second 301 computer would be advantageous because it would not only add greatly to our capacity but also give us a back-up facility.

With the two computers in operation, we were able to use one system entirely for composition. It provided straight-matter tape for both hot metal and photocomposition machines—Photon 513's which

we had the Photon Company build so they could be computer-driven.

The second 301 was—and still is—used for book-keeping and allied activities. This eliminates costly interruptions and reel changing under the original one-unit system when composition was needed.

That pretty well establishes the background. Now for more recent developments and plans for the future.

First, a Soroban Paper Tape Punch has been installed to replace our former punch in the primary composition system. This device has a speed capability of 300 characters per second as opposed to a maximum speed of 100 characters per second for the replaced punch.

The Soroban Punch presently is giving us an effective thru-put speed for all composition programs of 180 characters per second. As time permits, some modifications will be made to our production programs to capitalize further on the speed of the new punch.

Our original plans for installing a magnetic drum as a storage medium for the dictionary and the various programs now in use have been changed. Instead of a magnetic drum, we have two Data Disc Files which have a storage capacity of 22,000,000 characters each.

One feature of these devices is our ability to store in these an "Exception Word Dictionary," which is utilized by all composition programs. The dictionary is updated with all words once found to be erroneously hyphenated. As a result, all stations serviced by the computer center receive equal consideration by providing the proper names peculiar to their particular area or subject.

The Data Disc File should not significantly affect the thru-put speeds of any of our composition routines.

We have not abandoned the idea of eventually installing the magnetic drum as the primary storage medium. However, a final decision on that has been deferred until the anticipated delivery and installation of a whole new computer generation complex, the RCA Spectra 70.

Another piece of computer hardware has been installed on the supporting system. It is an RCA paper tape reader with 1,000 characters-per-second capabilities and a paper tape punch with 100 characters-per-second output speed. This replaced an original RCA combination paper tape reader-punch which had a transfer rate of 60 characters per second.

The thru-put speed of this system is approximately 80 characters per second for all composition routines.

As for our hot metal straight-matter composition, modifications are made from time to time, but the program is basically the same as we have had in use since December, 1962. This program currently is capable of handling any one of the 20 available fonts contained in the tables. Any font can be added to this program within a 24-hour period.

It has been a busy time for us in the production of straight-matter by photocomposition, using the American Type Foundry Model CS.

We have found the CS completely reliable for this production. Our program operates in the same manner and is capable of producing the same data as the hot metal program for any one of the five fonts available on discs.

We have installed six of the Model CS's at our smaller papers. Among the appealing features of the CS are its small size, relatively inexpensive costs, ease of maintenance and its consistent 10 lines per minute production.

While the CS satisfies immediate needs of our smaller papers, it is not capable of meeting our demands for high-speed computerized production in our larger plants. The answer to this problem appears to be the Photon 713, which is capable of 35 lines of straight-matter composition per minute. We have received delivery of 6 Model 713's.

Since the Model 713 is primarily a straight-matter machine, it will not replace the Photon Model 513's also in operation. Our current program for the 513 covers 15 different type styles and 90 special characters that can be delivered in any one of 12 sizes ranging from 6 through 72 points in variable line length up to 42 picas.

Another new development in conjunction with the Photon 513 is what we call a commercial disc, especially designed for composition of books, magazines, brochures and similar matter.

During recent months we have had a sort of three-way wedding of production facilities in our West Palm Beach plant. It involves the computer, the Photon 513 and the Perry Photo-Composer. The Perry Photo-Composer (Fig. 1) is an automated version of three manual ones we have had in use for several years. It is, in effect, an automatic full-page makeup machine.

Like Photon 513, the Photo-Composer takes punched tape from one of the 301's and this tape in-

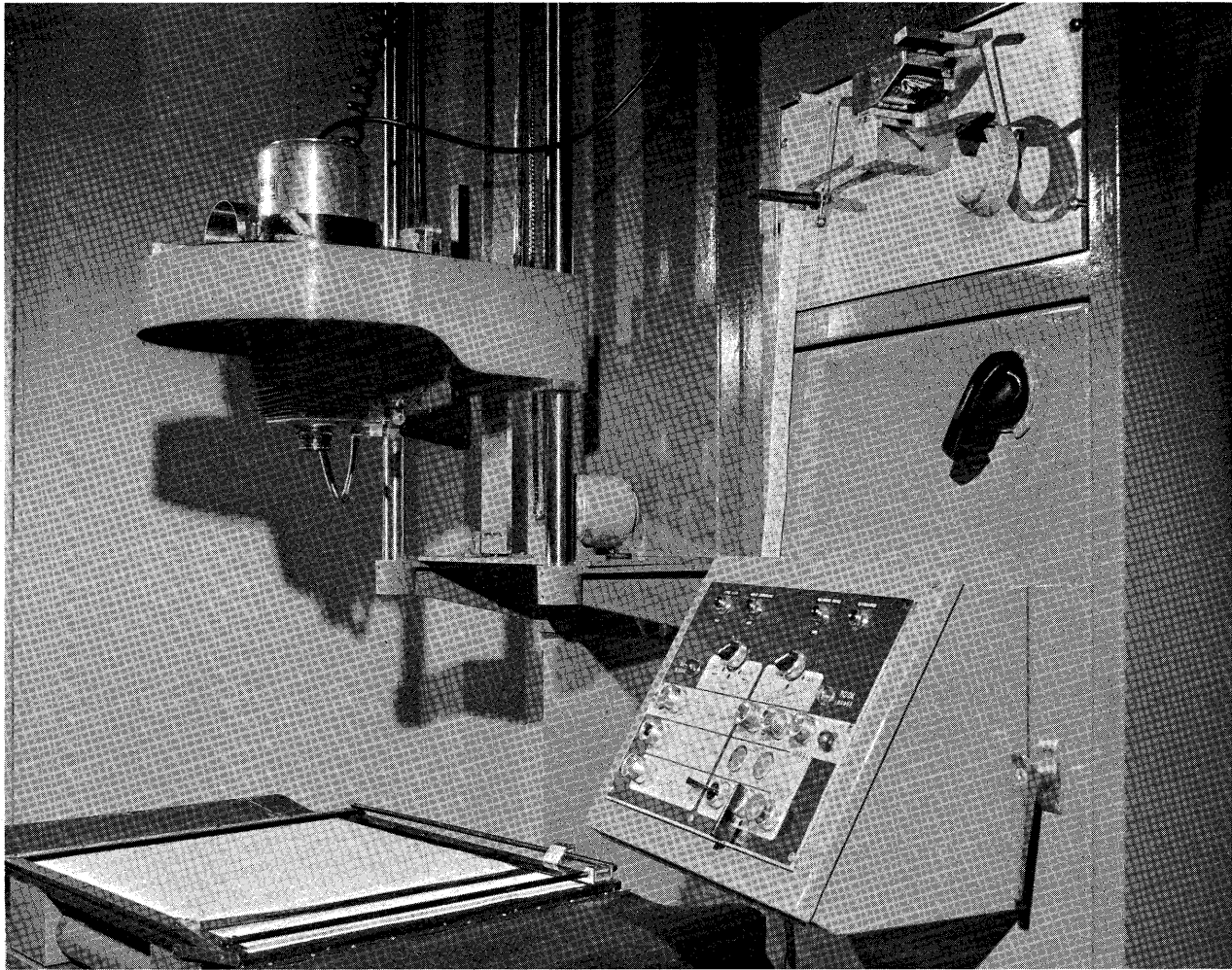


Figure 1. Perry Photo-Composer. Developed by Perry Publications, Inc., the Photo-Composer is an automatic page makeup device which positions and exposes type on film and adds, photographically, such features as pictures, artwork and borders. The Photo-Composer works with reversed film and automatically projects the image to the proper position according to directions supplied by paper tape codes.

structs its own computer to position and enlarge the film from the 513 to complete the automatic page makeup.

The machine is capable of automatically putting on borders, but does not add cuts and artwork. These two items are stripped in by artists in the quality control department.

At present, two programs are in operation for the Photo-Composer. The first is advertising block copy. This block copy, consisting of any mix required, can be delivered in a single exposure on the Photo-Composer, providing it is not greater than 30 picas wide and four and one-half inches deep.

The second is line copy. This line copy, consisting of any one parameter, can be delivered on the Photo-

Composer for any width up to a page, and any size depending on the enlargement requested, up to an actual limit of 8 times 72, or 576 points.

Let me give a step-by-step outline of how the Photo-Composer operates in producing what we feel are remarkable results.

Key to the Photo-Composer's functioning is a marked sheet of sensing paper (Fig. 2), measuring $8\frac{1}{2} \times 14$ inches. On this sensing sheet are 6,900 small printed boxes.

Each represents the lower righthand corner location of desired copy blocks to be called from the memory of the computer system by means of assigned code numbers.

After an ad layout is prepared, the sensing sheet is

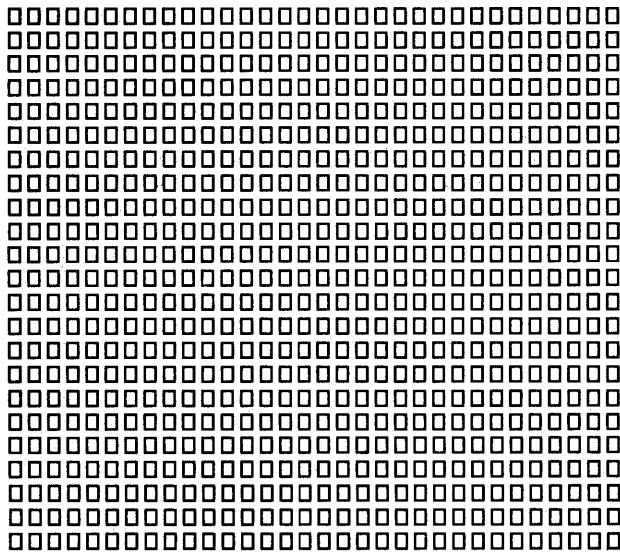


Figure 2. Portion of a mark-sensing sheet, key to operation of the Perry Photo-Composer. Each complete sheet contains 6700 boxes and measures $8\frac{1}{2} \times 14$ inches.

placed on the layout and the lower righthand corner of each copy block is marked, with pencil, on the sensing sheet.

When the marking of the sensing sheet is completed, the ad copy is then typed onto a Retina Reader typing sheet in the exact order and sequence as the marked sensing sheet.

Next, both sheets are fed into an Electronic Retina Computing Reader (which we shall describe in detail later). The Retina Reader produces magnetized tape that is, in turn, converted to justified output paper tape.

The tape for the copy then goes to the Photon, which turns out a negative of the desired copy. From there, the negative and tape produced from the marked sensing sheet are taken to the Perry Photo-Composer.

The tape moves the composer bed into proper position for the desired copy block which is on the negative. At this point, the composer pauses, photographs the block and then moves on to the next position.

The Photo-Composer is of extreme value to advertisers, who are able to have their predetermined ad style stored in computer memory and to indicate by parameter code on the copy block the exact format desired for reproduction and publication.

The parameter code is one of the secrets of the operation. This code is shown on the computer print-

out, so that all the advertiser need do is go through his catalog of previously used ads, select the copy block he wants to duplicate—and we do the rest, using new copy which he submits if he desires to change previous ad text.

The parameter code indicates type size, line length and style, whether type is to be quadded right, left, or center; whether there is to be leading between lines and whether copy is justified or line-for-line.

With the code, the adman can indicate use of all or just part of the copy block.

No text as such is stored; only the type sizes, line lengths, and type faces are stored.

To date, the Photo-Composer has been used only for ad preparation, but now we are experimenting with it on newspaper makeup.

I should like to tell you in some detail of the operation of the previously mentioned Electronic Retina Computing Reader.

The full name of the equipment is the Electronic Retina Character Reader System. It is built by Recognition Equipment, Inc., of Dallas, Texas. Electronic Retina is the trademark of that firm.

The Electronic Retina uses a two-dimensional matrix of photosensors that senses an entire moving character rather than a segment of it. The Retina has a character resolution approaching that of the human eye and reads up to 2400 characters per second. It eliminates sweeping and timing circuits and critical adjustments because spatial relationships are fixed by the retina itself.

One significant advantage of the Retinal concept over single-spot or columnar methods is that information regarding the entire character is transmitted simultaneously to the recognition unit in analog form. Other methods must store information until a complete character has been scanned before attempting to identify it, and this can be done economically only in digital form. Not only is some information inevitably lost in the conversion to digital form, but it is limited to a simple black or white; or black, dark gray, light gray, or white in a four-state system. The Retina, on the other hand, begins identification instantly and can recognize an infinite range of gray shades; this, in effect, adds a "third dimension" to the character.

Each photocell in the Retina matrix is influenced also by those surrounding it so that weak lines are emphasized and smudges eliminated.

The Electronic Retina equipment is fully compatible with the RCA 301 system.

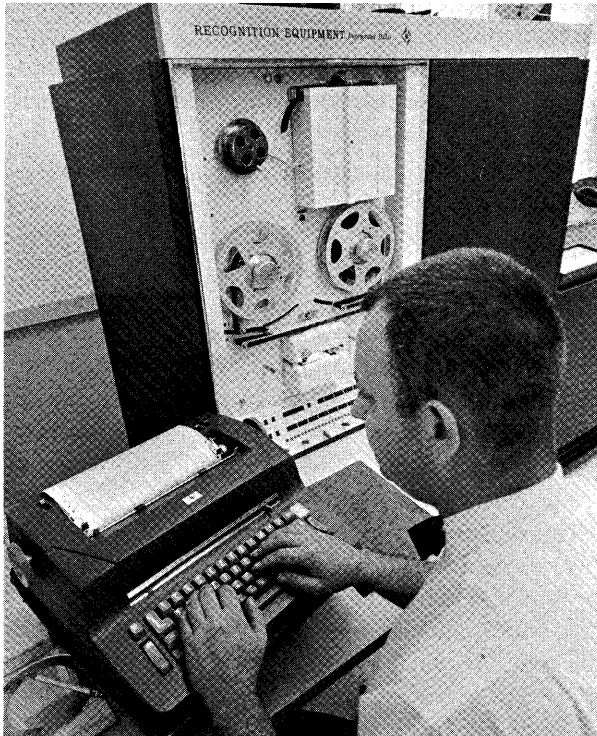


Figure 3. Retina Reader console.

In detail, the Electronic Retina system (Fig. 3) consists of: (1) a Rapid Index Page Carrier, (2) an Electronic Retina and Recognition Unit, (3) a Scientific Data System 910 Computer, and (4) two Ampex Magnetic Tape Stations compatible with RCA.

Hard copy is fed into the system and the characters are read by the Electronic Retina. This hard copy, prepared with special typewriters developed for us by Olivetti, can be editorial text matter, classified advertising set solid, semidisplay, display, legals, accounting data, or any other alphanumeric information.

The Electronic Retina converts the typewritten characters and spaces into predetermined electrical impulses stored on magnetic tape. The magnetic tape, in turn, produces justified paper tape to drive the linecasting machines—either hot or cold type.

The Electronic Retina will read printed or typewritten copy up to the rate of 2400 characters per second. That is approximately 28,000 average words per minute.

The machine processes pages which vary from 3 1/4 by 4 7/8 inches to 14 by 14 inches and in weight from 12-pound paper to 30-pound paper at a rate of from 10 to 30 pages per minute, depending on total infor-

mation to be read from any page. At any one time, different-sized pages, within the range just given, can be handled intermixed so long as there is no more than four inches difference in length among the pages in the batch.

The Electronic Retina closely resembles the functioning of a human eye. In fact, it was designed after a thorough study of the eye. It views constantly rather than intermittently. It senses black, white, and all shades of gray (Fig. 4).

The Electronic Retina also adds a slight amount of "jitter" when viewing a subject—just as the human eye does to smooth out the edges of rough lines.

Overall, the Electronic Retina is capable of:

1. Producing unjustified magnetic tape for the 301 to set all local display advertising, using the present 301 programs.
2. Producing unjustified magnetic tape to set all classified display and set solid classified advertising, using the 301 programs.
3. Producing unjustified magnetic tape to handle all accounting, using the 301 programs.

```
PW NEA GIFTS 2A 4-25 abcdey tues
:((f;080$110;:nea gifts-4-25(M);:simple bridal gifts(M);(a
gifts(M);:
(M)NEV YORK (NEA) - Very personal giftsa gifts ba for a bride
are remembered through the years. One such present that fits
this special category is the bridal handkerchief.(M);:
(M)A crisp, white, beautifully edged hand hanky tucked within
a sleeve or pocket of the bridal gown is an age-old tradition.
Bjtsia But in this aa era of action with little thought such
an item often is forgotten.(M);:
(M)The gift has lasting meaning primarily because it can be handed
down through the family. Afer a fa After isa its use in the wedding
ceremony, the hnakyscan hanky can be turned into a christening
bonnet for the first child, taken apart for the marriage of
the child years later.(M);:
(M)Select a richly laced hankechief and include instructions
for turning it from hanky to bonnet and back in the
package.(M);:
(M)The makers of Niadara starch suggest this method of making
a bonnet for baby: Select a 11- to 12-inch handkerchief. Use
spray starch and iron if crisp and sa square. Fold handera
handkerchief in half and fold in half again so a square a quarter
of the original size is formed. Slipstitch folded edges
together.(M);:
(M)Maintain a double thickness and open the square out to form a
point. Bring point down 11/2 inches along stitched edge to form a
triangle and ba tack in place.
This forms a basic the hat spaa shape. For further shaping,
backstitcha backstitch two 1/4-inch darts about 2 inches long,
11/2 inches from either aa side of center back. Finish with ribbon
rosettes or bows and narrow ribbon ties. Since all is hand-sticheda
hand-stitched, merely clip the thread to return hanky to original
form.(M);:
```

Figure 4. Copy ready for the Retina.

4. Reading and transmitting unjustified magnetic tape to the 301 for all text matter programs.

We were able to get magnetic tape stations compatible with the 301 tape stations. As a result, we are able to go from magnetic tape to magnetic tape through taking, by hand, magnetic tape from the Electronic Retina and positioning it on the 301 tape station by hand.

After typed copy is read by proofreaders and editors, the copy is corrected before reading by the Electronic Retina by using a special correction routine.

If there is any one central point about the Electronic Retina in which interest centers, it probably is in this matter of corrections and how they are accomplished (Fig. 5). Since the production routine calls for input of typewritten copy, editorial people, quite naturally, wonder how the penciled editing changes are reconciled. On the other hand, production people are concerned about proofreader marks and how corrections, long the major bottleneck in photocomposition from the standpoint of speed and cost, can be made quickly and economically.

The key to this routine is making the corrections before composition is accomplished. Or, rather, making the corrections before the Electronic Reader provides the idiot tape for the 301.

Here is how it works:

Each line on each typewritten page of hard copy is identified by number. Editing and proofreading changes are marked on the hard copy before it is inserted in the Electronic Retina. These changes and corrections are marked in the conventional manner on regular copy and, in addition, an identifying mark—usually numerical sequence—is made in the right-hand margin at the end of the affected line.

When the editors and proofreaders complete their work on the copy, the original is returned to a typist at one of the special typewriters.

The typist then retypes the lines, or segments of lines affected by the changes, using a new piece of copypaper as a correction sheet. The correction sheet then is fed into the Electronic Retina ahead of the original copy which is to be corrected.

The reader stores these corrections and its coded instructions concerning them in its memory system. Then, as the regular text comes in behind the correc-

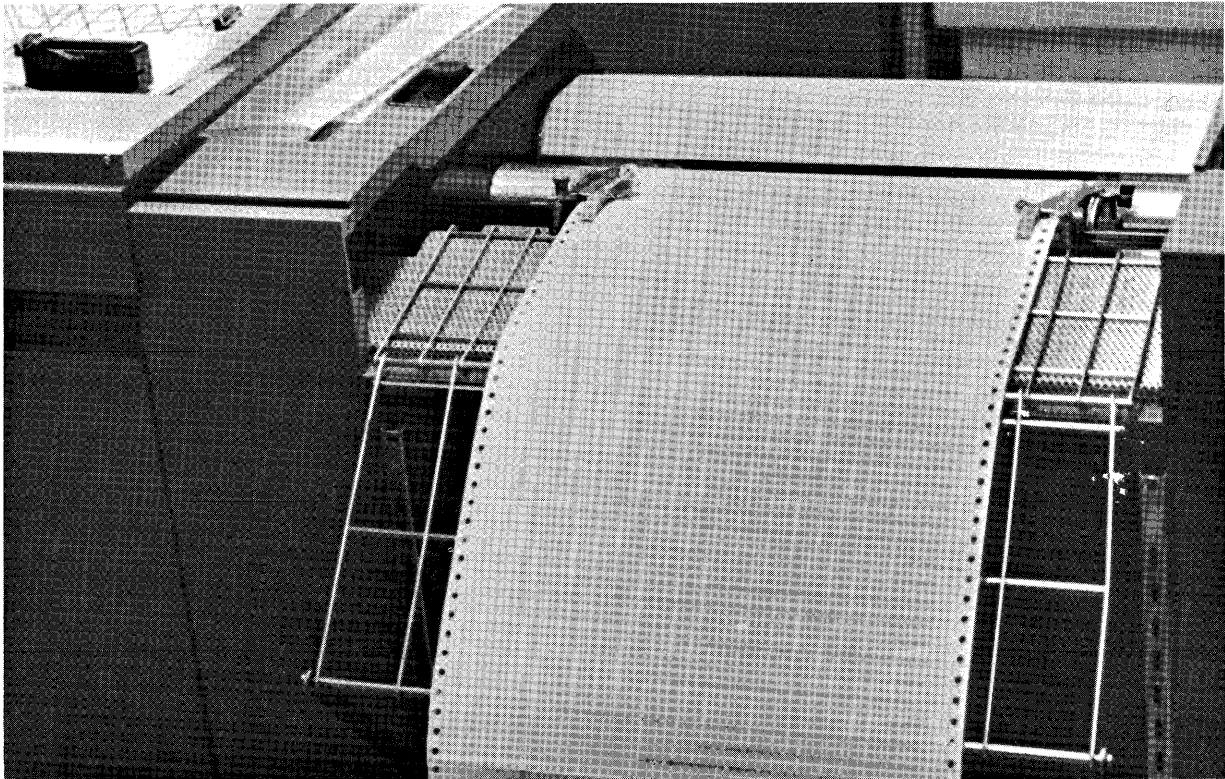


Figure 5. Computer printer and tape station.

tion sheet, the Electronic Retina's computer determines whether there is a correction in each line.

If there is one, the Retina wipes out the original copy and accepts the corrected line, or lines, inserting the revised material until the correction is completed. Under this routine, the Retina will accept corrections ranging from transposition of two letters in a single four-letter word to insertion of dropped copy ranging up to an indefinite number of lines. Then it returns to the regular text, accepting it without interruption until another corrected line is encountered.

We spent considerable time training typists to assure us an ability to produce accurate typewritten copy for the Electronic Retina. And our programmers labored over a long period programming the typewriter face—or font—from the Olivetti typewriters.

At the same time, we have been at work on an advertising insertion order for national, local, classified, legal and accounting.

The insertion order contains the code number for accounting, production, classification, and the number of the product, as well as all other insertion instructions and other information. It also enables us to transmit the exact information from the advertising insertion order to the actual account. If it sets solid classified, it sends the composition to the composing room, and it counts the number of lines of copy so that we can have the exact line count for makeup purposes. When the last ad is stored in the computer, we know exactly how much space to allot for the classified advertising section.

One of the major problems confronting us before all our properties can make full use of the Electronic Retina is the matter of getting hard copy from the various points into the Retina at West Palm Beach. Several possible solutions are under study, most involving broadband transmission systems.

A partial answer to this problem can be supplied quite simply. It involves installation of one or more of the special typewriters at the individual points for use in preparing hard copy for insertion orders, accounting information, editorial features and other such matter. The material then would be sent to West Palm Beach by mail. After processing, the justified tape needed for production at the point involved would be transmitted back over our Data Speed network, while the other material needed in the West Palm Beach center would be retained there.

Now, what about the economies possible through use of the Retina Reader?

I should like, in this regard, to quote Mr. Herman L. Philipson, Jr., president of Recognition Equipment, Inc.

"The economic consideration of optical character recognition," he tells us, "hinges on two very important factors: the cost of a keypunch stroke and the cost of an error." You can figure typical keypunching costs roughly as follows, he estimates:

	<i>Monthly Cost</i>
Salary	\$380.00
Vacation (2 weeks/year ÷ 12)	14.62
Sick leave (3 weeks/year ÷ 12)	21.92
Insurance, payroll taxes, pension	53.00
Personnel Total	\$469.54
Supervision and G&A (20% of salary)	\$ 76.00
Floor space	25.00
Overhead Total	\$101.00
Equipment rental	\$ 65.00
Card stock	31.50
	\$ 96.50
MONTHLY TOTAL	\$667.04

Cost per keypunch stroke formula:

$$\frac{\$667.04}{\text{month}} \cdot \frac{1 \text{ month}}{140 \text{ hrs.}^*} \cdot \frac{1 \text{ hour}}{8,000 \text{ keystrokes}}$$

$$= \frac{\$667.04}{(140)(8,000)} = .06\text{¢ per keypunch stroke}$$

This table shows the average cost to be about .06¢ per keypunch stroke, or 6¢ per 100 strokes with no verification. If you have 72 keypunch operators producing 2 million cards per month (average 40 characters per card), monthly keypunching costs with no verification are approximately \$48,000 per month.

Recognition Equipment's Electronic Retina Computing Reader, which leases for approximately \$15,000 per month, is capable of processing up to 13 million card equivalents per month in single-shift operation, assuming 75% operation efficiency.

Thus it is clear that directly reading information with current OCR equipment offers a tremendous profit potential over keypunching as a data input method. What is less clear is the fact that, by simply typing this information that cannot be read directly

* Based on 21 days per month, 6.67 hours per day.

instead of keypunching it, the user can realize immediate, substantial savings while he implements direct-read applications that realize the true profit potential of optical character recognition. Consider what Mr. Philipson offers as typical typing costs:

<i>Monthly Costs</i>	
Salary	\$380.00
Vacation (2 weeks/year ÷ 12)	14.62
Sick leave (3 weeks/year ÷ 12)	21.92
Insurance, payroll taxes, pension	53.00
<hr/>	
Personnel Total	\$469.54
Supervision and G&A (20% of salary)	\$ 76.00
Floor space	12.50
<hr/>	
Overhead Total	\$ 88.50
<hr/>	
Equipment rental	\$ 10.00
Paper and ribbon	17.00
<hr/>	
	\$ 27.00
<hr/>	
MONTHLY TOTAL	\$585.04

Cost per typing stroke formula:

$$\frac{\$585.04}{\text{month}} \cdot \frac{1 \text{ month}}{140 \text{ hrs.}} \cdot \frac{1 \text{ hour}}{14,000 \text{ keystrokes}^*}$$

$$= \frac{\$585.04}{(140)(14,000)} = .03¢ \text{ per typing stroke}$$

This table shows the cost to be .03¢ per typing stroke or 3¢ per 100 strokes. To produce 2 million card equivalents per month (72 operators and \$48,000 with keypunching and no verification) would require 40 typists. The total input cost, including rental of an Electronic Retina Computing Reader, would be approximately \$39,000 per month.

Probably of more significance than the cost of a keypunch stroke is the cost of an error or the cost of reentry. The cost of reentry varies from \$.20 to \$10, depending on the application.

The Electronic Retina Computing Reader presently is reading with a substitution or error rate of less than 1 error per 100,000 characters. To achieve this degree of reliability with keypunching would require 100% verification.

* Effective stroke rate used for comparison only. Based on increase in stroking speed plus reduction in number of strokes by eliminating the need to stroke leading zeros, duplicated information, etc.

Early in 1967, we may receive delivery on the RCA Spectra 70 computer complex.

Preliminary examination discloses that this hardware, in lieu of our present 301's, will give us a capability 28 times that of the present setup. Besides the enormous increase in speed, error will be reduced because tape punching operations in present systems will largely be eliminated.

The reason we are considering this elaborate, complex system is our belief in the necessity for a systems approach to the whole problem of producing a newspaper. Just how costly all this eventually will prove, I don't believe there is any accurate way to figure at this time. We simply are considering it because we believe it is the only correct approach to the subject in our particular case.

The functions and capability of the Spectra 70 are not considered separately, but rather as an integral part of a total system. We expect the total system not only to do hyphenation, justification, bookkeeping, billing, ad composition, classified and circulation, but also market research.

The Spectra 70, in basic design, is a total management information system. It has the hardware, the software, the language, the communications, the total capability to meet our needs now and for years to come, we believe. This it can do on line and in real time. The Spectra 70 is a true third generation computer. It works on a monolithic integrated circuit. With it we may even do foreign-language encyclopedias.

Since Spectra 70 harmonizes with most computers, it will save our present programming investment and, we hope, eliminate costly reprogramming, retraining, and reinvestment. Its speed is measured in nanoseconds; our present 301 systems operate in millionths of seconds. This will be an important step ahead in our crucial dictionary retrieval system.

We have been working on the cold type process continuously since 1946 and are completely sold on it. The only bottleneck is the need for a higher speed straight-matter machine. We believe a solution is near, either by installation of Photon 900 with its 150 lines-per-minute speed or an Electro Gun or Page Generator with speed in the neighborhood of a full, complete page in less than 200 seconds.

Another vital function which the Spectra 70 can perform as a total management information system is to provide what we call a Business Profile Analysis.

We would program into the memory bank a wide

range of pertinent information concerning each of our properties. This not only would include data regarding actual operations of the property, but significant details about the specific community and area, such as population, growth trends, degree of circulation penetration by households, employment, industrial expansion, degree of competition from other media, and so on. Current operating data will be combined with this basic stored-up information at regularly designated intervals—probably every seven days—and management will be provided with a detailed performance report far more elaborate and informative than conventional operating statements in use today. These reports not only would indicate trouble areas immediately after they develop, but would show trends and indications to enable management to head off trouble before it develops.

In more conventional typesetting operations, the computers' function is this: Copy is keyboarded on TTS machines producing six-channel tape at the rate of 800 newspaper lines per hour.

Punched either at the West Palm Beach plant or at any of the 25 outlying plants in our group, this unjustified tape is sent to the main plant via high-speed Data Speed (telephone) relaying lines.

In West Palm Beach it is justified and hyphenated by the 301 logic system, with the exception word dictionary, and returned via the Data Speed lines.

Each Perry plant has its own composing and press equipment.

And so, as you can see, we at Perry have been caught up in a quick-paced march of progress toward better, faster, more profitable operations for ourselves and improved services and products for our customers.

An increasingly major part of our business is the audio-visual field.

Not long ago, a major university was encountering trouble with an audio-visual book. This book contains listings of films that most major colleges have in their libraries. It needed to be referenced into two sections in two different manners: first, the alphabetical listing of the films and, secondly, listings according to subject.

The biggest problem was that in order to produce a film catalog, the film director or the school needed to create for the printer both an alphabetical listing in the front section as manuscript, and then to sort down all these films and place them in a subject-heading file for the printer. This was laborious, time-consuming and, in most cases, conducive to error.

The university usually spends about six months producing a supplement to the master catalog. By the Perry method, we produce the catalog in two weeks after receipt of manuscript.

As an input, the computer uses from the university a card into which all the information is typed into specially prepared blocks. These cards, after receipt at the Perry organization, are typed onto Olivetti sheets. These typed sheets are fed into the Retina Reader and stored on magnetic tape. The magnetic tapes are then placed on the computer.

The magnetic tapes are read through the computer producing a computer printout. This print is sent as a sheet to be read and corrected by the school, which runs down to film listings, checks our copy, and returns a proofread computer printout. This then is turned over to the typist, who makes any corrections, deletions or additions. At that point these are fed onto a magnetic tape, which at a later stage is merged with the original tape, making a third or corrected magnetic tape.

During the whole process the keynote is that we are only printing out memory we have, not setting type, or pulling proof, or in any way going into the preparation of typesetting.

When the printouts have been approved for typesetting, at that moment the button is pushed, and we begin to spew out justified paper tape sorted into alphabetical order or numerical order as the school requires—and this tape then is ready automatically to run photo typesetting machines to produce the film listings in alphabetical sections.

The computer program not only drives out this tape but also paginates while delivering the paper tape. Thus when the photo composition comes off the machine, it is paginated and need only be applied to the grid sheets for the artist rather than be cut up and pasted. Time is saved, plus the fact that many persons can work on the unit at once—rather than have one worker paste and cut.

The second section, or the subject heading index, in the computer program will go back into the original input and find in a particular coded field all the subject indexes in which that particular film applies, creating a section in alphabetical order. It will take these films and place them in order, listing the film titles. This film may appear in more than one category in several locations. Once this has been done by the computer, another tape is dumped and delivered to the Photon which created the subject-heading index.

In the event the school wants subsidiary report checks, these can be made. As an example, if a school wants to find out how many films by a certain producer are currently in the school film library, the computer goes down this magnetic tape and picks them off. Or if a particular division of the university wants a list of all films that run less than 15 minutes—this can be obtained. Or if a school of science or of engineering so desires, we can pick out of the master listing a special supplement pertaining only to that school. There is no costly manuscript by the school; there is no manuscript submitted for the subjecting indexes.

Schools for which we have done work include the University of Michigan, Michigan State, Wayne State University, the Dade County School system in Miami, and Florida State University. In some cases, input preparation time and the cost of clerical help have been reduced 30 to 40%.

Big savings occur the second year, when new manuscript need not be submitted, but only changes and deletions to the existing catalog or the stored magnetic tape. Thus as the year progresses, the school simply keeps up with new film by filling out a card and dropping it into a box. At the end of the period, the cards are collected and shipped to us. We need not even reread what is on tape as it is corrected. Thus a laborious proofreading task and chances of error are eliminated. These new cards are put in separately and a special printout for the school's checking comes out.

Assuming there are changes in 10% of the final listings, or 4000 film titles, the time needed to get the first justified tape is only a matter of hours. Pages start going back to the university the first week. With regular typesetting methods, the usual time period is six months or more.

Due to publicity we have received on this audio-visual system based on magnetic tape, many cataloging firms have contacted us to work with them on a method of creating catalogs from year to year by storing past material that has not been changed, thus saving typesetting, storage of metal and the usual fumbling at the last minute for a complete catalog ready for the printer. Some of the firms are large and we have done pioneer work for them, in either developmental activities or on actual jobs, so that they can benefit from the Olivetti typewriter and Retina Reader.

The automated approach is particularly interesting to one automobile manufacturer who produces a

reference manual and a repair manual. In the past, they had to farm this job out to five different typesetters.

The problem was that if the typesetter who had chapters 1 through 10 fell behind, this held up complete collection of the chapters. Much time was spent by company personnel to check the printers at their various stages. Checks had to be made on quality, compatibility of type facings, corrections, etc. This was quite a problem.

In our case, we are able to mass many machines under one organization and produce photocomposition rapidly—using instant retrieval of programs in memory storage. We are now in the process of creating a large general-purpose catalog program that would encompass almost every aspect of cataloging. In six months to a year this master catalog program will be ready to take on almost any style or collection in retrieval, or catalogs in general. Currently we are limited to the audio-visual program, which enables us to use up to eight available blocks for storage and retrieval plus a ninth block for corrections.

So far as the newspapers' future is concerned, we hope one day to use the computer to evaluate and qualify news in order to provide the editorial content of the papers with proper balance.

An editor can quickly review a news story from the printed copy and assign each story a category, which would be encoded for future computer use.

Additions or corrections could also be encoded; and because a computer can digest and arrange 12 hours of normal copy in roughly 15 minutes, the editor is freed of the humdrum of sorting his own news.

All original news stories would be translated from the paper tape on to a magnetic tape, which would then act as a memory. A second tape would be prepared, listing all corrections, additions and deletions.

Each news item would be prefaced by a category number and a priority rating. A third tape is then prepared, carrying the editor's instructions. This tape contains the news balance formula the editor is using as well as the size of the "news hole."

Inside the computer, all news could be assigned to categories, such as: government and politics; war, defense, diplomacy; economics, business and travel; crime; public moral problems; health and welfare; accidents and disasters; science and invention; educa-

tion, classic arts, religion; popular amusements; mass media; and general human interest.

After determination of the size of the "news holes," the emphasis for each category and allowable length for each story, here's what would happen:

1. Category percentages would be multiplied by the news hole figure to determine desired lines per category.
2. Memory tapes containing all available news would be searched, lines counted and stories selected.
3. Desired lines per category would be revised to reflect surplus or deficiency.
4. Desired lines per story would be set according to editing formula and story priority.
5. Available story line would be matched with desired amounts.
6. Results would be revised to reflect surplus or deficiency in any particular category.
7. Highspeed printer would produce copy arranged according to categories with each line one news column in width.

One advantage of such use of the computer would be to fortify the newspaper against charges of being unfair. Most everyone at some time has felt the value content of a news item was unbalanced—and frequently this has been so. We expect the computer could prevent any such future injustice.

As for use of the computer in market research, there is interest on the part of advertising agencies. Most major agencies want a tremendous amount of detailed information to back up dollar expenditures and, in fact, a number offered to set up a computer center which would, in cooperation with member newspapers of the American Newspaper Publishers Association, establish a National Central Advertising Plan.

The central office machine headquarters would be connected with all agencies and all newspaper members. All information concerning the newspapers would be fed to the machines, which would process the data for use by the agencies. The agencies would send orders to this central office as well as requests for information. The central office would, in turn, send orders to the newspapers and do billing for the newspapers.

It is possible the office also would send checks to the newspapers on a weekly basis.

The agencies could in reality depend on this central office for virtually everything except creative work. We should capitalize on the electronic marvels embodied in the computer complex, not only to enable us to achieve greater profits but to render greater service to our subscribers.

The functions of a newspaper are many: to make the community's economy work through effective advertising; to permit expression of public opinion through letters to the editor, so that all sides of issues can be debated; to make the community aware of any bad situations into which it has drifted; to acquaint community leaders with activities of other leaders; to help the reader understand his environment so that he may crusade for its improvement; to strengthen the moral resolutions of citizens; to provide a medium of entertainment; to attend to the small wants of classified advertisers; and to give readers a sense of identity.

All these functions can be performed better and more economically through the computer.

Currently we are at work on a computer program designed to help us build better and deeper submarines—another activity of Perry Publications, Inc. We produce a somewhat famous small submarine called the "Cubmarine," one of which recently helped locate the missing hydrogen bomb off Spain. With computers, we can test in a matter of hours new designs that normally would take weeks of trial and error.

If all stress formulas, weights, balances, movements and other stability factors are put into the program and we decide to enlarge or shrink a displacement, we can find out almost immediately what will happen to all the other factors involved. Therefore, it becomes possible to find an optimum point at which to stop with very little extra expense.

We have also been in the television business. We owned and operated Channel 2, NBC affiliate for Central Florida with studios in Daytona Beach and Orlando. There is a place for the computer in such operations, too.

We have devised a program to keep logs, availabilities and other factors and automate the station's operations. The tremendous amount of information that goes back and forth between an advertising client, agency, national representative, and TV station

and its market can be enormously speeded up and assimilated by proper computer programming.

At Perry, 1965 was the year when the experimental era with computers ended and they began paying off handsomely. Determination of this fact was made by the increasing use each of our outlying publications made of the central facility in West Palm Beach.

Late in 1964, we had set up a pricing schedule which charged each user so much per minute of computer facility time. Individual use of the facility was entirely voluntary. And as the papers began to measure their use against this dollar cost, their increasing use made it obvious the hump had been overcome. The total percentage use of the computers jumped beyond 75% overall. In some peak time areas it jumped to 100%, using both 301 systems.

This has resulted in our setting up a new pricing schedule very much on the order of a television or radio network. Prime time has a higher price than, for example, the midnight to 6 A.M. period. Now our managers are rescheduling their computer requirements to best take advantage of the new lower rates at less prime time.

Perry Publications had estimated it would take about three years for the initial sizable investment in computers to pay off. This timetable has been met. Spectra 70 could, we feel, pay off in even less than three years.

With Spectra 70 we foresee its use to explore various theories of decision-making. The computer can help us with an analytical approach to business problems by having the decision-maker break down large, complex problems into a series of relatively simple ones. Thus, we should be able to exercise judgments and preferences on each of the relatively simple problems and allow the theory to indicate the

most consistent action in the larger, more complex one.

An extension of this approach also will allow the business manager to evaluate the desirability of purchasing additional information before he acts. When added information is obtained, this approach will indicate the proper combination of that data with the decision-maker's previous judgment.

The old concept of waiting six months to a year for an outside audit to tell you whether your business is headed in the right direction has become a critical hazard. Before computers, for example, we never received our monthly profit and loss statement before the 15th of the following month. Today we get it on the fourth of the following month. Soon we will be able to get a weekly profit and loss statement each Monday morning.

Before computers, our production level and employee load were comparable with other operations of our size in our industry. Today, at *The Post-Times* in West Palm Beach, with the aid of computers, we are producing approximately 50% more work than we were six years ago with approximately half the production personnel we then had. Our total employment is up, however. We have more personnel in the editorial department, for example.

The jet airplane, direct distance dialing, concepts of instant checkless banking, and computers working at nanosecond speeds are giving business and industry a realization that time itself has tremendous value. Twenty years ago, it took an hour or more to engrave a printing plate. Today, quick etching can do it in a few minutes. Tomorrow, electron guns and lasers will do a whole page within seconds.

While the cost of all these tools themselves will be enormous, the cost of installing them will be even more enormous. Industry must have profits to afford them. But the results will be worthwhile.

**A
SPECIAL PURPOSE COMPUTER
FOR
HIGH-SPEED PAGE COMPOSITION**

Constantine J. Makris

*Mergenthaler Linotype Company, Division of Eltra Corporation
Brooklyn, New York*

INTRODUCTION

For several years now, computerized page composition has attracted the interest of workers in the printing and data processing fields. Considerable work has been conducted in the past in both fields by various concerns with the primary goal to speed up the laborious work of page makeup and thus reduce the page makeup cost.

Our studies in the page composition area have shown that page makeup can be implemented by either one of two methods—namely, page composition employing the stored program flexibility of a general-purpose computer or page composition executed by a computer specifically designed for that purpose.

We adapted both methods, for each one in itself presents attractive features depending upon the application and the nature of the matter which is to be processed.

This paper discusses only one phase of the Linotron System; specifically, the typography program of page formatting as executed by a special-purpose computer called the "Page Formatting Computer."

During the system concept period, the philosophy

of this system design was guided by three primary objectives:

1. High-page composition machine throughput.
2. Flexibility of machine to handle a large variety of tabular and text formats.
3. Machine simplicity to insure smooth man-machine interface.

The first two sections of this paper present the Linotron System and its organization in the page typography program and a functional synopsis of the characteristics of the phototypesetter.

The succeeding five sections discuss the organization of input material to the page typography program and present various definitions and problems encountered in the process of page formatting.

The last section presents the internal organization of the page formatting computer and describes and brings into focus the major characteristics of the system.

LINOTRON SYSTEM

The Linotron (a Mergenthaler-CBS Laboratories development) is a high-performance phototypesetter

with x - y plotting features that produces high-quality lexical-graphical matter under the control of a properly programmed magnetic tape.

The formatted tape which drives the Linotron phototypesetter is the product of the typography program performed by either a general-purpose or a special-purpose computer.

This discussion is focused on the manner with which the Page Formatting Computer (PFC) is organized and tied in with the overall page composition and typography system.

Figure 1 shows a simplified diagram of the Linotron System. The basic task of the PFC in the overall Linotron System is to receive from magnetic tape edited lexical-graphical matter. Then the PFC by means of its computational capability properly composes and sets this matter on the page according to prescribed page formats of good typography. The system outputs this formatted information on a magnetic tape medium with all the necessary instructions to drive the Linotron Phototypesetter.

THE LINOTRON PHOTOCOMPOSER

Figure 2 shows the block diagram of the Linotron Photocomposer and the flow of data through the system.

It is conveniently subdivided into seven large functional subsystems:

1. Magnetic tape recorder
2. Tape control and buffer storage
3. Control logic
4. Character generator
5. Cathode ray tube display system
6. Positioning servo
7. Recording camera

Magnetic Tape Recorder

The magnetic tape read transport reads the control tape which is the precipitation of the page formatting program. On this tape are recorded codes of the characters to be typeset with the manner with which they will appear on the copy. In addition to the character codes, character positioning and system control codes have been properly recorded to insure proper functional operation of the system. Data can be furnished to the system in either 6, 7, or 8 level codes at 556 bpi.

Control Logic

This is the section of the machine which reads the information from the input core buffer and sorts the character codes from the positioning and/or control codes. It routes the character codes and character size codes into the character generator block; and the positioning codes into the positioning regis-

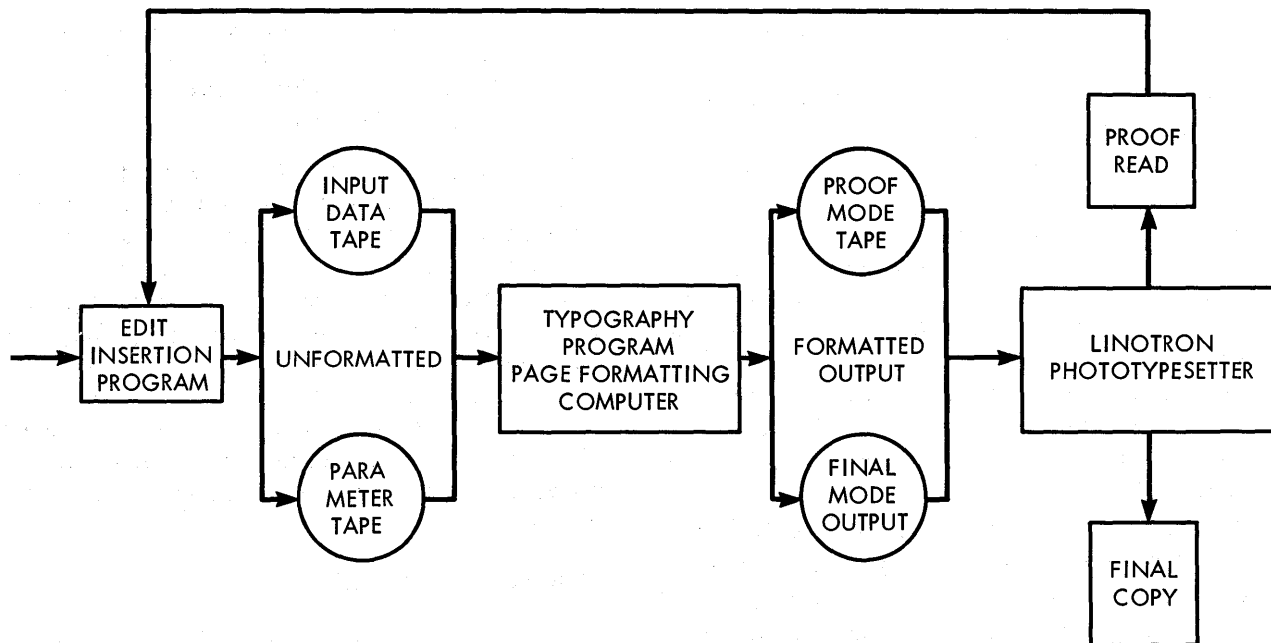


Figure 1. Simplified block diagram of Linotron system.

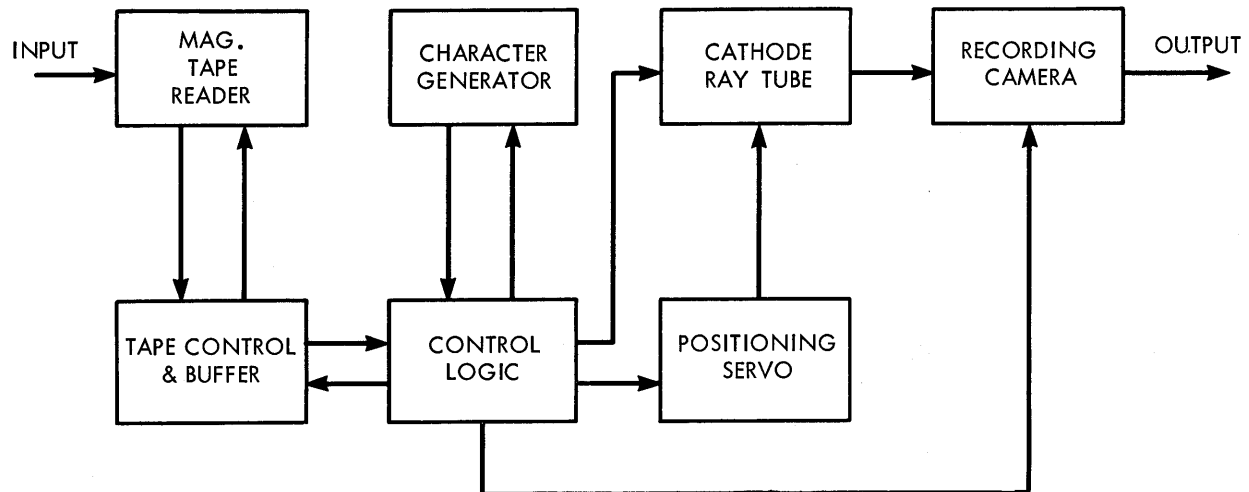


Figure 2. Block diagram of Linotron Phototypesetter.

ters. It also generates control signals from the servo block, cathode ray tube assembly and recording camera assembly upon the recognition of control codes.

Character Generator

The function of the character generator is to receive the character and character point size code and in return generate a high-quality video signal of the character. The character is selected from an assembly of 1024 characters whose typographical configuration is stored in four film plates (grids).

The heart of the character generator is the Linotron tube—a single-envelope, high-quality tube which transforms the light image of characters focused on its photocathode into a character video signal of a given size.

The generation of the character video signal is accomplished by optically focusing the entire complement of a character grid (256 characters) into the photosensitive cathode of the tube. The resulting electron image beam from the photo cathode is imaged onto an aperture plate by means of magnetic lenses and then wobbles back and forth at the aperture plate. A small aperture of 0.001 inches is associated with each of the 256 character images at the aperture plate. As the images are wobbled back and forth, the aperture's scanning of the image is accomplished as in the image-dissector video camera tube.

The electrons which pass through the apertures are all trapped except for the aperture that scans the selected character. This is accomplished by means of a bar matrix (16 vertical \times 16 horizontal) that

lines up with each aperture and is biased with negative voltage. The control logic places positive voltages on the selected x - y bars that line up with the desired aperture. The electron beam in the form of pulses is amplified by means of the multiplier section of the tube thus producing a video signal.

Cathode Ray Tube

The cathode ray tube assembly consists of a high-resolution, high-intensity CRT and deflection system. The electron beam of the tube which produces a spot of less than 0.001 on the screen sweeps into synchronization with the scan rate of the Linotron tube. The character video signal is utilized for unblanking the Z axis of the tube thus the output screen exhibits an accurate reproduction of the original character. Size of the characters can be changed by changing the amplitude of the sweep. This is controlled automatically by the control logic when a point-size change has been called for by the tape.

Positioning Servo

The precise character positioning on the screen face of the CRT is accomplished with the aid of the positioning servo, which makes use of a second CRT, the spot position of which is accurately referenced to a set of precision-ruled gratings. Initially, the horizontal and vertical coordinates of character position are obtained from the control logic and used to position the spot close to the correct character position. Then the servo circuitry brings the spot precisely to the desired position—the precision gratings

being used as a reference—and keeps it there until the character is printed. The CRT display, which is actually producing the character, is slaved to the positioning servo tube. Its beam accurately follows the beam motion of the servo tube. Thus, the positioning of the character is accurately controlled by the precision servo.

Recording Camera

The full composed page displayed by the CRT is photographed onto the film or paper by the recording camera. The film of the camera advances frame by frame (page length) as instructed by the logic control block when it recognized film advance code.

FORMAT ANALYSIS AND DEFINITIONS OF PAGE FORMATTING

A study and analysis of a large variety of page formats was a prerequisite to the design of an efficient page-formatting system. This study disclosed the following fundamental page typesetting characteristics:

1. The formatting program of any new page format does not necessarily have to be rewritten from scratch.
2. The page makeup program consists of a number of distinct typesetting operations, such as line justification, column justification, etc. Each typesetting operation required in the page makeup can be split into a number of small independent routines, each one of which is programmed to accomplish just a fragment of the overall program of the typeset operation.
3. The program of any page makeup can be accomplished by the proper collation of independent typeset microroutines.
4. Each page format contained its own format parameters which have to be inputted and stored as program parameters.
5. The editing process could be minimized to just a simple data item labeling.

The following page formatting definitions had to be developed as format analysis was taking place:

1. *Format*: The unique spatial layout of the page items (such as columns, fields, paragraphs, etc.) within the page, and the specific sequencing of the typesetting routines within the item in the page as well as the specific sequencing

of the routines between one item on the page and another item on the same page.

2. *Typesetting Routines*: Format-independent, self-sustained routines which implement a defined fraction of a typeset operation program.
3. *Format Program*: The program which sets the complete page format by means of selecting and sequencing the proper typesetting routines.
4. *Format Parameters (Program Parameters)*: The spatial parameters of the format (such as, X, Y marginal dimensions, line measures, etc.) and typographical parameters of the format data (such as typeface, point size, etc.)
5. *Edit Instructions (Page Item Identifiers)*: Labels inserted before page items which identify copy blocks of identical spatial, typographical and typesetting characteristics within the format.

Figure 3 shows two pages of a representative text format.

CLASSIFICATION OF INPUT DATA

The source data input to the PFC has been classified as (1) edited lexical-graphical input data and (2) page format parameter data.

Appropriate format parameters had to be defined for each format on hand which had to be applied to the input and stored as program parameters.

The format parameters have been divided into two categories: (1) the typographical parameters of the format that prescribe the style of type and size for each page item and (2) the format dimensional parameters. Since the format parameters had to be changed occasionally at the command of the typographer or page format designer, they had to be generated separately and stored in a separate tape—the parameter library tape.

It is noted that the bulk of the typographical format parameters consists of the character widths for each typeface at different point sizes. For example, for 1024 characters that the Linotron Phototypesetter handles simultaneously and for 8-point sizes, there are 8192 character widths with a byte of 3 BCD digits maximum. Assuming 2 BCD digit storage per core locations in a character width look-up

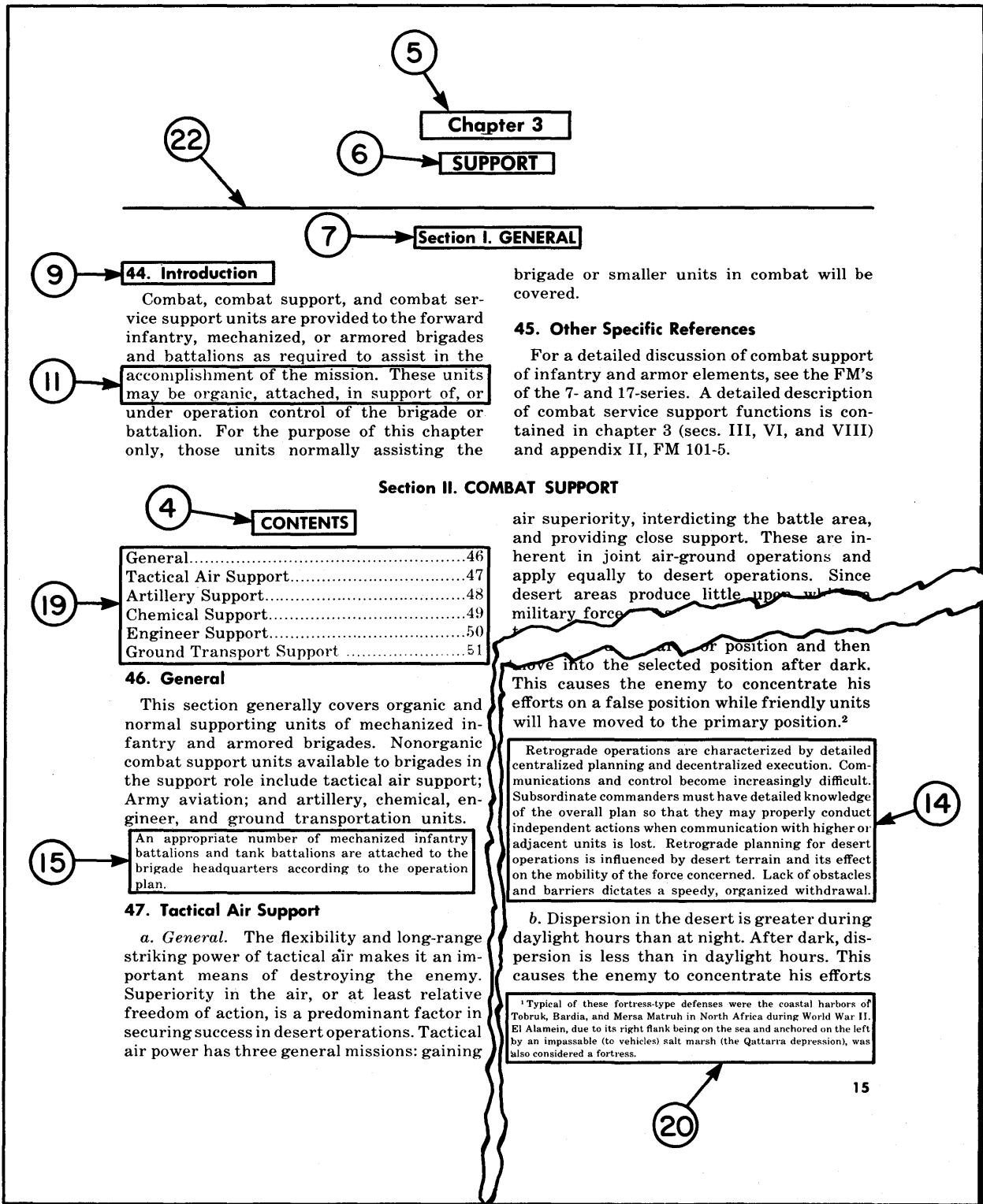


Figure 3. Representative text format.

table, there is a 16,384 of maximum core requirements plus time consumed by the program to address and retrieve the character widths.

To avoid this core storage and time requirement, the character width parameters have been separated from the parameter library tape and a flexible circuit logic block has been designed which would accommodate the character width parameters with fast access and minimum cost.

ORGANIZATION OF INPUT EDITED DATA

The edit program of the system is the program where raw data together with edited instructions are generated directly from the keyboard. The purpose of this program is to provide properly edited, unformatted data on paper tape or magnetic tape. This data is utilized as input to the formatting program.

The edit instructions have been made simple in order to allow some flexibility to the keyboarding. A decimal number from 1 to 99 inserted before the data of a page item (copy block) serves as the edit instruction. Copy blocks which demonstrate fixed formatting characteristics such as quad left data, quad right data, main text justified copy, headings, etc., are labeled with the assigned item identifier instruction number. Table 1 shows a list of page copy blocks that have been assigned to an item identifier edit instruction number.

ORGANIZATION OF FORMAT PARAMETERS

The typographical page parameters fall into three categories: (1) type face of copy block, (2) point size of characters, and (3) character grid number where the type face is located.

The dimensional parameters of the page format have been subdivided into the horizontal and vertical parameters as follows:

Horizontal

1. *The X_{cn} group*: These are the absolute horizontal vectors that define the horizontal location of various copy blocks on the page with the left side of the page as reference, such as absolute margin of column, indentations, etc.
2. *The LL_n group*: These are relative dimensions that define the line length of copy blocks useful in line justification operation.
3. *The S_{min} and S_{max} group*: These parameters define the maximum and minimum range of values that

Table 1. Assignment of Copy Blocks to Item Identifiers for Format

Copy Blocks	Item Identifier Numbers
Chapter number (single column)	01
Chapter title (single column)	02
Section number and title (single column)	03
Contents (title of table of contents)	04
Chapter number (double column)	05
Chapter title (double column)	06
Section number and title (double column)	07
Side head (quad left)	09
Side head (quad right)	10
Body text	11
Text (with shorter line length than body text and having same right-hand margin)	12
Text (with shorter line length than body text and centered to text body)	13
Text of lower point size (with same line length as body text)	14
Text of lower point size (with shorter line length than body text and centered to text body)	15
Notes (with same line length as body text)	16
Notes (with shorter line length than body text and centered to text body)	17
Notes (with shorter line length than body text and having same right-hand margin)	18
Table of contents (single column leadered table)	19
Footnote	20
Line plot (single column)	21
Line plot (double column)	22
Underscore	23
Beginning of caption	24
End of caption	25

For Tabular Formats

Fixed fields and/or unjustified line copy (line that does not overset)	01-17
Justified line copy (line that may overset)	18-20
Constant page heading	21
Running heading	22
Column heading or subheading	23
Notes	24
Running heading (index)	25

the word space can receive in the line justification procedure.

4. *The Δx group*: These are incremental fixed values selected for a particular format to position on page copy blocks with respect to previously set data.

Vertical

1. *The Y_{cn} group*: These are absolute dimensional parameters that reference and position items from the top edge of the page.
2. *The Y_{LA} group*: These are the incremental parameters that define the line advance of text copy.

3. *The Y_{LP} group*: These are parameter values that define the various incremental separations between the end and the beginning of a copy block or vice versa, such as end-of-text paragraph to the following heading.

4. *The Y_o or ΔY group*: These values define the limits and tolerance of the vertical data settings in column justification operations.

For tabular formats where the fields of data and digits per field of data are fixed, parameters besides dimensional ones have been derived—such as, number of fields, number of digits per field, number of lines, number of columns, number of groups of lines.

HYPHENATION

In setting a justified line of type of small line measure, hyphenation is unavoidable; hence, the system had to be equipped with a hyphenator algorithm that could hyphenate the overset word and thus help the program justify the line.

The hyphenator algorithm that has been developed by Mergenthaler is based on the eight-window principle developed by Lockheed Co. It consists of a logic mechanism that recognizes and samples the structure of the word as the word flows through a field of 11 windows (a 6 bit \times 11 position shift register).

Each character of the word under investigation is shifted from one window to another at the rate of the applied clock. The contents of each window are decoded and the information is supplied in a parallel form into a cluster of hyphenation logic rules. The logic of each hyphenation rule consists of a static gating. When the contents of the windows have the proper character combination, which can satisfy either one of the rules, then a hyphen can be inserted by the control section of the Hyphenator between the characters that surround the checkpoint.

The Hyphenator has demonstrated high accuracy—over 97%—with acceptable hyphen efficiency on representative vocabularies. It has been built in a modular form with a data transfer rate of 0 to 100,000 characters per second, suitable for interfacing high-speed computers.

MAJOR CHARACTERISTICS OF THE PFC

The basic characteristics of the page format processor are as follows:

1. The capability to compose a large variety of tabular and text page for-

mats by receiving edited source data on magnetic tape.

2. The ability to format and process pages by means of a wired-in program at a considerably higher throughput rate than the conventional stored program computers of medium size and cost.
3. The ability to change the program of the machine conveniently by merely changing either one or two printed circuit matrix cards depending upon the format program complexity.

INTERNAL ORGANIZATION OF THE PFC SYSTEM

General

The Page Formatting Computer that can prepare formatted tape to drive the Linotron Photocomposer is basically a special-purpose computer with a wired-in rather than stored program. It can be viewed, however, as a general-purpose page formatting machine in that the program of the Processor can be changed readily with different page programs. This is possible by plugging into the computer the desired format program control boards which actually are matrix logic cards.

The simplicity and flexibility with which the format program of the machine can be changed is attributable to the internal organization of the system.

The control of the CPU of the system consists of a library of independent typesetting routines whose program has been specifically designed to handle microtypeset operations. The proper selection and sequence of these microtypeset routines by the format program control can compose a variety of tabular and text formats. The formatting program is accomplished at relatively high speeds due to the parallel operation of the arithmetic computations and the simultaneous execution of the routines program.

The machine is equipped with three core memory units:

1. *Input Memory* (buffer)
8K \times bits
2. *Output Memory* (page memory)
16K \times 6 bits
3. *Working Memory* (scratchpad and parameter storage)
2048 \times 17 bits

The number system with which the system performs arithmetic operations is in BCD. Five BCD digits parallel arithmetic is performed in two arithmetic sections that operate simultaneously.

The internal coding system of the computer is based on 6-level BCD code. The system, however, can accept input data recorded on either 6-level or 8-level code, in BCD ASCII or extended BCD coding system.

The computer is equipped with a cluster of address tracking, index and utility registers.

The machine control section of the processor responds to nearly 160 command instructions which can be addressed by the format program control independently.

The basic machine cycle (MC) of the system is 5 microseconds and instruction can be executed from about 0.2 MC to 10 MC depending upon the nature of the routine called for by the instructions.

Figure 4 presents the system's internal organization. It is subdivided into three units: (1) processor storage unit, (2) central processor unit, and (3) input/output unit.

Processor Storage Unit

This unit includes the input memory (IM) and the output memory (OM) of the system.

The unformatted data is stored in the IM. The CPU operates on the data stored in the IM and forms lines. When the CPU has formed a line of type in the IM, it transfers it to the OM.

The OM is the page memory of the system. In this memory, the data of a complete page is formatted and assembled.

In certain formats—such as double column formats—the last item on the page, as recognized by the CPU in the IM, may cause repositioning of all page items already stored semi-formatted in the OM.

The OM, therefore, stores the data of the page until the last page item is read from the IM which may or may not overset the page.

Central Processing Unit

The various functional blocks of the central processing unit have been designed and developed with the modular concept in mind. These blocks have in-

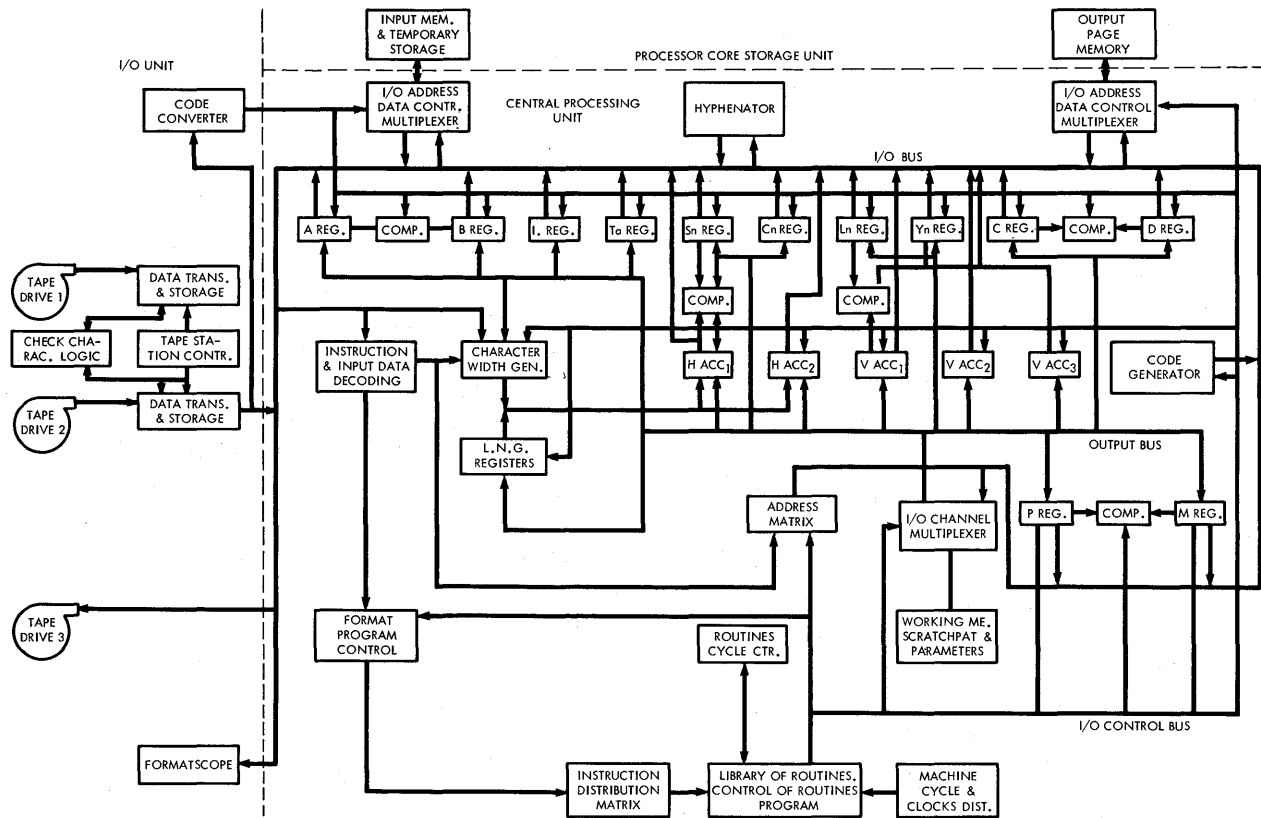


Figure 4. Page formatting computer system diagram.

dividual controls of their own, thus making them independent, self-sufficient and asynchronous. Under the control of the routines, the blocks can be time-shared to perform the routine program in a synchronous manner with the rate that the routine itself dictates.

The CPU is divided into the following sections:

1. The working memory section
2. The arithmetic section
3. The address, index and utility registers
4. The input instruction and data decoding section
5. The character width section
6. The library of control program of type-setting routines
7. The format program control
8. The hyphenator

The Working Memory Section

The working core memory of the system is basically the scratchpad storage of the CPU and the format parameter storage.

During the page formatting program, the results of temporary computations are stored in this memory.

Words up to 17 bits long can be retrieved or stored in this memory practically from all the registers as well as from the accumulators of the arithmetic section in either parallel or sequential mode.

This is made possible by means of the input output address and data channel multiplexer that can communicate with the rest of the blocks. The typographical and dimensional parameters of the format are also stored by the parameter library in a predetermined area of this memory.

The Arithmetic Section

The arithmetic section of the system has been divided into two units—the horizontal arithmetic unit equipped with two accumulators that can perform parallel addition and subtraction of 5 BCD digits in 1.5 microsecs.; and the vertical accumulator unit equipped with three parallel accumulators of the same speed. The separation of this section into two arithmetic units and the reinforcement of each unit with several accumulators insures fast and parallel computations of the horizontal and vertical arithmetic equations involved in setting the data on the x - y plane of the page. Comparison decision modules and auxiliary registers—namely, S_n , C_n , L_n and Y_n registers—have been incorporated in this section to speed

up, facilitate and simplify the routines program. All accumulators can store their results in either the scratchpad or input and output memory through the I/O channel multiplexing.

Address, Index, Utility Registers

Addressing, tracking and indexing of data in the scratchpad input and output memory is implemented by means of the respective registers.

The A-B registers are primarily utilized by routines that form the line in the input memory. The I_0 register is utilized by routines to help sort out graphic data, footnotes or any type of headings from the input data and subsequently store this data temporarily in a separate area of the input memory. Registers P and M are primarily utilized by routines to index in the scratchpad memory the addresses of lines and various items that lie semi-formatted in the output area as well as the temporary storage area of the input and scratchpad memories. Registers C and D are associated with indexing data address of items already stored in the OM.

Input Instruction and Data Decoding

This is the section that decodes the edit instructions and data characters when the machine is in the read input data mode.

It sorts out the input edit instructions (Item Identifiers or Typographical Control Codes) from the printed data codes and keeps up to date the format program control by means of control signals on the state of instruction and data the program is working on. Certain routines receive control lines from this section.

This block provides command control signals to the character width block and P_z , F, G storage registers when input character codes are to receive a width value. It also provides command control signals to the address matrix of the working memory to switch the memory to the appropriate core area where the parameters of the particular copy block are stored.

Character Width Section

The function of this block is to receive a 6-bit character code from the input/output channel multiplexer of the input memory and yield as an output the typographical width of this character in parallel BCD form.

The character width that is being outputted by the

block is a function of the three typographical parameters which are received from the parameter section of the working memory or directly from the input; namely, the typeface, the point size of the face, and the Linotron grid number. The width information is accessed at its output in less than 300 nanoseconds thus speeding up a typographical command so repetitive in printed matter. The character width information is routed via the program of the specific routine into the Horizontal Arithmetic Unit. This block is rather flexible since it holds up to 16 different type faces which can be substituted externally by a different one when the job calls for it.

The Hyphenator

The duty of this section is to receive the word which oversets the line and output the same word with all the grammatically legitimate hyphens that it can find. The overset word is transferred from the IM into the Hyphenator by means of the I/O channel multiplexer and from the Hyphenator into the scratchpad area of the working memory. When the program finds that a word is split within the justification zone, then the line is set in the (OM) with the last word hyphenated.

Control of Typesetting Routines Program (Routine Library)

Like any other control section of the computer, this is the instruction execution section of the CPU. It consists of 160 independent routines subdivided into the following four distinct sets:

1. Routines associated with the horizontal computation and set up of text body copy blocks in the page memory.
2. Routines associated with the setting and temporary storing in the input memory copy blocks that may cause a column or a page to overset, such as headings, footnotes, nonmandatory graphics and captions as well as updating and storing page-derived running heading copy blocks.
3. Routines that are associated with the vertical setting of copy blocks within the columns of the page.
4. Routines associated with the setting of table data, generating and setting page number and setting initial parameters within the computer blocks.

When the routines are instructed to perform, they control the operation of each individual module within the CPU and thus determine and synchronize the direction and flow of data within the machine. They output clocked signals to the computer blocks that they control by having access to all clocks that the machine cycle and clock distributor provide. Interfacing control signals are received by this section from various blocks of the CPU for certain routines whose program calls for it.

Upon the completion of their program, the routines turn themselves off and the majority report the end of the routines program to the Format Program control which initially activated them.

Figure 5 shows the flow chart of a typical program for such a routine.

The Format Program Control

The format program control is the controller of the central processor unit in that it controls the

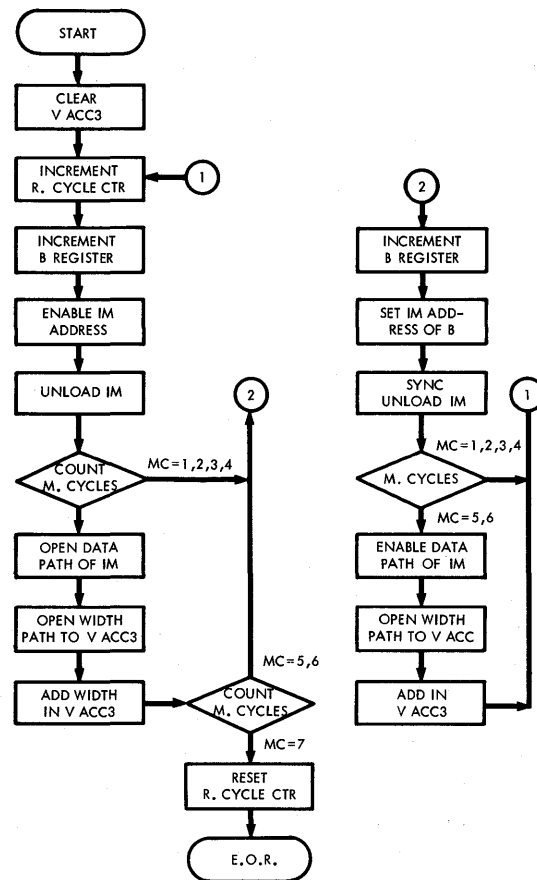


Figure 5. Flow chart of routine scanning graphics data in IM.

various computations and operations which produce a properly composed typographical page.

Upon the provision of edit instructions and data by the decoding section, the format program control selects and sequences the appropriate typesetting routines.

As a program controller, it supervises the state of the routines and the state of the page being formatted. It follows prescribed page composition rules on graphics insertion, footnote formatting, running heading updating, etc. Following is a list of some of the page make-up rules which it obeys and executes:

1. A double-column nonmandatory graphic* referred to on a page will be set by the program at the top of the next page.

2. A single-column nonmandatory graphic referred to on a page will be set by the program either on the same page or on succeeding pages. The program will never place a graphic in a column preceding the graphic reference in the text.

3. A single-column nonmandatory graphic that is referred to in the first column and the program discovers that this graphic cannot fit in that column will be placed at the top of the second column. The program will continue processing the text in the first column until the valid column depth is reached.

4. When a single-column nonmandatory graphic appears in the second column of the page and the program discovers that it cannot fit it in that column, it will transfer this graphic to the next page on the top of the first column. The program will continue to process the data in the second column until the valid column depth is reached.

5. Any single-column mandatory graphic † will be placed by the program right after the reference point in the text. If the graphic does not fit in the first column, the length of the first column will be left short. The program will transfer this graphic at the top of the second column and the depth of the second column will be made equal to the depth of the first.

6. Any single-column mandatory graphic that is called for in the second column of the text and the program cannot fit into this column will be transferred to the top of the first column of the next page. The program will readjust the depth of the second column to equal the depth of the first column.

* Nonmandatory graphic is the graphic which is not necessarily set on the same printed page.

† Mandatory graphic is a graphic that is placed right after the point of text reference.

7. Any double-column mandatory graphic that appears in the first column for the first time will cause the program to split the already processed column into two parts. If the split occurs between a heading, the heading will be part of the second column and the second column will be longer than the first. If the split occurs between text, the program will justify the second column to the depth of the first column. The program will continue processing the data following this graphic in column one.

Exceptions to the rules are as follows: (1) A minimum depth will be required of the first column in order to be able to justify the second column to the depth of the first. (2) If a split occurs between a heading and the heading is the first line of the column, then the column will not be split. The second column will contain no text and the graphic will be placed after the text in the first column.

8. If a double-column mandatory graphic is called for in the second column, the material preceding it will be arranged by the program into two justified columns above the graphic. If the graphic does not fit, the program will end the page and the graphic will be placed on the top of the next page. If in the process of splitting the column, the program senses that the column is split immediately after a heading, the heading will be placed in the second column and the first part of the column will be justified to the second part of the column.

9. If a graphic-text format contains double-column mandatory graphics and a single-column nonmandatory graphic, then the single-column nonmandatory graphic will be treated in the program as a double-column nonmandatory graphic. Rule 1 will then apply.

10. If a footnote reference appears in the text of the first column being processed, the footnote information will be placed by the program at the bottom of the first column. If the depth of the footnote data is such that the program cannot fit it into the first column, then the program will place this footnote at the bottom of the second column.

11. If a footnote reference appears in the text of the second column being processed, the program will place this footnote at the bottom of the second column. If the depth of this footnote is such that the program cannot fit it in the second column, it will place it in the following page. The footnote reference (superscript) will be an asterisk instead of a numeric in such cases.

12. If a single-line heading appears at the end of the column being processed and the program cannot fit this heading with one line of the text following, then the program will transfer the heading to the top of the second column.

13. If a new-page-derived running heading is detected, the program will replace the wording of the old-page running heading which is temporarily stored with the new one.

14. Any double column heading that appears in the first column will cause the program to split the already processed column into two parts and the program will proceed as in Rule 7.

The Format program control section can house up to six different format programs which are plain matrix boards that can be plugged into the machine.

The Input/Output Unit

The Input/Output Unit is equipped with three tape stations and a page display monitor—the Formatscope.

The first and second tape stations can accept tape drives reading either IBM 729 tape formats or IBM 360 tape formats depending upon what tape drive is chosen by the input requirements. Both tape stations can accept tape drives with speeds up to 112.5 ips reading tapes with packing density up to 800 bpi.

Thus the system accepts data recorded in either 6 or 8-level code. The code converter block reduces the 8-level code to 6-level BCD code, the internal coding of the system. Either one of the two input stations can be utilized for input data loading or parameter loading.

The third tape drive is utilized as output when the formatted page stored in the output memory is to be loaded on a magnetic tape.

The Formatscope is a video display output device which, as an x - y plotter, upon command can display on a video screen the data of a complete formatted page. Each character is shown as a dot. It serves as an on-line diagnostic device for the operation of the machine proper.

COMPUTERIZED TYPESETTING OF COMPLEX SCIENTIFIC MATERIAL

J. H. Kuney, B. G. Lazorchak, S. W. Walcavich

*American Chemical Society
Washington, D. C.*

and

D. Sherman

*Inforonics, Inc.
Maynard, Mass.*

The successful development of computerized typesetting has been a process of striking a balance between input coding and capacity to program the variety of decision-making steps which characterize the typesetting process. The most frequent of these decisions—justification—has proved most adaptable to computer handling. Even hyphenation, with its variety of rules, has been handled via the computer with varying degrees of success. But composition for scientific journals creates special needs both for input coding and for computer processing in the handling of non-text elements such as special characters, tables, mathematical expressions, and graphic data in the form of chemical structures. Spacing considerations and character selection become much more complex and involve many decisions which are a matter of choice, not rule.

In this paper we will describe a macro coding system for inputting text, mathematical expressions, and tables, employed in a computer-based typesetting system which performs routine typesetting calculations and also implements the variety of decisions made by an input operator in the setting of scientific

material.¹ First tests of a new program for the typesetting of chemical structures will be detailed.

Macro expressions are used extensively to instruct the computer programs to call into operation a defined series of instructions and/or to recall textual strings. There are two distinctive operations within the macro mode—initialization and recall, either of which may be put into effect at any moment in the input stream.

The system contains both permanent and temporary macros. The permanent macros apply to the repetitive format patterns which characterize the typographic style of a particular journal. Temporary macros are used to meet specialized needs within a particular processing period. For example, if a long chemical term is to be used frequently in an article, it may be entered as a temporary macro and thus save keyboarding time. The keyboard operator has available all of the permanent macros entered in the system and may himself create temporary macros to meet the particular needs of the moment. In effect, he uses the macro codes as a typesetting language to reduce input strokes and to optimize computer

processing. The macro code system is the basis for the development of a formalized typesetting language now under development at the ACS.

INITIALIZATION AND RECALL

The operator enters the macro mode by punching a code which is the color shift key. The operator follows the color shift key with a control key to denote initialization and one more key to identify the type of macro to be initialized. An alphanumeric or special character macro identification code used in recalling the macro is keyed next. The operator then enters the series of typographic functions and/or text matter that will define the particular macro expression. A second color shift code terminates the macro mode. When the computer system senses a macro initialization mode, it compiles the subsequent macro expression and allocates it to a macro library accessible from disk storage. Each journal in the system is allocated 40 permanent macro positions of 90 characters each, and space is available for 80 temporary macros of 90 characters each. Alphabetic and special characters are used to identify permanent macros and numerics are the keys to temporary macros.

MACRO USAGE

Once a macro is initialized by the following coded sequence: color shift, control key, i, ID, content of macro, color shift—the computer compiles the content and stores it on the disk. The macro may be accessed at any point in the stream of input data by the following calling sequence: color shift, macro ID, color shift. At this point the computer accesses disk storage and merges into the output stream the previously compiled macro expression.

In tables, the macro serves to facilitate the setting in columnar form and is initialized with a slightly different coding sequence: color shift, control key, t, ID, content of macro, color shift. Here the character "t" instructs the program to process the macro to handle columnar spacing using the tab key to separate columns within the macro expression. Each tab key begins the definition of a column in the table. The definition usually consists of column length, justification within the column, and other relevant typographic functions or text matter.

Figure 1 is the hard copy of the Flexowriter input to the computer system. The initialization of the macro takes place on the second line with the punching of the "&t!". The macro which follows contains

the line lengths of the respective columns and instructions for centering the material of the respective columns within the given line length. The tab key is used to separate column entries. The operator may call in other macros or alter justification for any given entry as might be required with automatic return to the original macro control. Disengagement from the tabular mode is automatic when the text is completed or when the operator punches an end-of-line code. The tab key reenters the tabular mode under macro control. Figure 2 shows the table as set on the Photon under control of the computer-produced tape.

SETTING MATHEMATICAL EXPRESSIONS

In the setting of mathematical expressions, the spacing considerations and the variety of character selection become so complex that we have attempted little more than to reduce the spatial relationships to some linear form for input coding. An actual example will illustrate how macro expressions can be used in the setting of mathematical formulas via computer as well as the variety of decisions the operator must set for the computer. Hopefully we would expect to work toward a method of handling which would transfer more of the decision-making choice to the computer.

In setting the equation shown below, four lenses and seven fonts were used as follows: normal 8 pt., numerator 8 pt., denominator 8 pt., and 18 pt. for the two-line characters; italic, greek, roman, math, caps and small caps, superior, and inferior fonts.

$$\delta I' = \epsilon_n \left[\int_0^L \left(\sum_{j=0}^N v_j \left| \frac{\partial f_j}{\partial y_n} \right| \omega_n \right) dl \right] \quad (21)$$

First, the operator must determine that the equation will fit within the assigned column width of the journal. Next, the equation is divided into vertical segments wherein the spacing treatment is the same. Since we can accommodate three levels of typesetting in each line, this particular equation causes no special problems on the horizontal level.

Next the operator must determine the line length of each segment. This can be done in several ways:

1. Calculate by setting on Photon and reading off a counter;
2. Estimate on chart of frequently used combinations;
3. Process by computer for calculation.

Table III. Distribution of Notation Size (4383 Compounds of Commercial Deck Assigned Wiswesser Line Notations)

Designation	Size	Cumulative	Designation	Size	Cumulative
1	1	1.0	D	14	89.6
2	2	1.8	E	15	91.3
3	3	10.4	F	16	92.9
4	4	19.2	G	17	94.1
5	5	29.5	H	18	94.7
6	6	39.6	I	19	95.4
7	7	56.8	J	20	95.9
8	8	60.3	K	21	96.4
9	9	67.8	L	22	96.6
10	10	74.5	M	23	97.0
11	11	80.0	N;NZ	24-36	100.0
12	12	84.3			
13	13	87.5			

Figure 1. Friden hard copy of table input to computer.

Since many of the combinations which appear in equations are repetitive, the operators become skilled in estimating segment lengths. The respective segment lengths are added, subtracted from the over-all line length, and one half the difference becomes the measure of the lefthand indention in order to obtain a centered condition of the equation. The sixth segment is not included in this calculation since the key number sets flush right on the over-all column width. The remaining five segments are to be set as follows: (1) flush right, (2) center, (3) justify, (4) center, and (5) flush left.

The operator is now ready to begin punching the input tape for the setting of the equation shown above. His next step is to write the instructions using temporary macros to control the spacing of the segments. The first macro controls the setting of the line of text preceding the equation and contains justify instructions, font, size, leading, and line

length. These are shown in Fig. 3. In the setting of the equation the operator used the macro for segment (1) and the remaining instructions controlled the following steps: greek font, italic font, roman, greek, inferior, 18 pt. lens, half set, 8 pt. denominator lens, inferior, zero set, 18 pt., math, no flash, half set, flash, normal set, 1/16 space, 8 pt. numerator lens, superior, 18 pt. lens, math font. In a similar fashion the remaining segments were key-punched. After processing in the computer, the output tape was run through the Photon to produce the result shown in Fig. 3.

TYPESETTING CHEMICAL STRUCTURES

The problems of setting chemical structures have been considered in the overall context of how computers process data which consist mainly of graphic information. One approach is to develop notation systems which employ a linear sequence of letters

Table III. Distribution of Notation Size (4383 Compounds of Commercial Deck Assigned Wiswesser Line Notations)

Size			Size		
Designation (S)	No.	Cumulative %	Designation (S)	No.	Cumulative %
1	1	0.1	D	14	89.6
2	2	1.8	E	15	91.3
3	3	10.4	F	16	92.9
4	4	19.2	G	17	94.1
5	5	29.5	H	18	94.7
6	6	39.6	I	19	95.4
7	7	56.8	J	20	95.9
8	8	60.3	K	21	96.4
9	9	67.8	L	22	96.6
0	10	74.5	M	23	97.0
A	11	80.0	N-Z	24-36	100.0
B	12	84.3			
C	13	87.5			

Figure 2. Table as set on Photon after computer processing.

and symbols to express graphic data. Generally this approach entails the use of an elaborate and somewhat unnatural set of rules to translate graphic information into a non-graphic cipher system. Another approach is to describe rather than translate graphic information within a system of topological coding by showing atom-to-atom connections in a compound, with the atoms as nodes and the connections as branches of a network. In the ACS system all of the input data is preserved, including orientation and proportion. Chemical structures are mapped directly onto a graphic matrix and processed as purely graphic data.

The use of a computer to assist in typesetting chemical structures is an example of tying together complex graphic input and output devices. The current inputs to the system are paper tapes produced by typewriters specially designed to produce readable chemical structures and coded paper tape. Two such

typewriters are in use, and a third is in the near-final stage of development.²⁻⁵ Input tapes for the initial tests of the programs for setting chemical structures were supplied from the Army Chemical Typewriter operated by Herner & Co. for the Walter Reed Institute for Army Research.

The basic system output is a paper control tape to drive a phototypesetting machine. An IBM 1460 computer has been programmed to accept input tapes from the Army Chemical Typewriter, to process and transform the input data from linear strings to a graphic matrix, to analyze the grid-form structure, and finally to punch a paper tape to control a Model 200 Photon equipped with an eight-channel reader and a specially designed matrix disk containing the components required to build chemical structures. The further ability of the system to generate topological connection tables is envisioned in later development.


```
&Etest,j000,d000,g1,p,bl&o&o&o
/1g&j&3/9&,100&346
/11&z&r&170
/12&z&c&355
/13&z&c&385
/14&z&c&422
/15&z&r&256
/16&z&r&346
```

} MACRO INSTRUCTIONS

Equation 17 will reduce to

$$\delta I' = \epsilon_n \left[\int_0^L \left(\sum_{i=0}^N v_j \frac{\partial \bar{f}_i}{\partial y_n} \omega_n \right) dl \right] \quad (21)$$

At the optimal value of I' , when the extremal is not restricted, the value $\delta I'$ must vanish for all variations $\omega_n(l)$. It is therefore seen from Equation 21 that a necessary condition for an optimum is

```
ⓐEquation 17 will reduce to&x150&o
①;ϕd;#I&3 ;ϕe;=n &l/3$/rT/4;=/dO/3&l;F/rT&f/s/x/5&5L/3&l'
②/8w②/5&5N②/4;=j 0
③/8;ϕu;=j
④/5;$2/d9/s;#f;=j④/8&3/f&m/f&m/f&m/s/r④/4;$2;#y;=n
⑤/8;ϕo;=n/3&l&m/8;#d1/3&l=
⑥/8&3(21)
&x150
```

```
ⓐmat the optimal value of &t3I, when the extremal is not restricted, the value ;ϕd;#I&3 must vanish for all
variations ;ϕo;=n&3(&t3l).It is therefore seen from Equation 21 that a necessary condition for an optimum is&o
&e
```

Figure 3. Hard copy of input for setting of mathematical formula. Temporary macros (circled) control positioning of formula elements. The computer-set formula produced from input is shown in the inset.

INPUT DATA PREPARATION

The paper tapes produced by chemical structure typewriters consist of normal alphanumeric symbols, special symbols to represent chemical bonds, and X, Y coordinates which supply a frame of reference for the input data strings. Coordinates are punched automatically by the typewriter when the spatial reference of the input is changed by rolling the platen in either direction, tab, backspace, carriage return, etc. The X, Y coordinates refer to the location of the beginning of the string of symbols preceding the platen movement, backspace, etc.

The chemical structure typist need make only the decisions necessary to reproduce accurately the structure provided as copy. The Army Chemical Typewriter has three cases in order to provide a large symbol set. In addition, the typewriter provides certain options to simplify the typist's job; these are: nonspacing keys, double length extended keys (to make a bond and vertex with one keystroke) and keys which print a high or low version of a symbol (for example, upper dot or lower dot; upper and lower here do *not* refer to case). Thus in typing /, the typist would use the lower dot, diagonal, and

upper dot keys without having to roll the platen to type the diagonal or the second dot.

Errors in layout or alignment during the course of typing a structure are frequent and allowable. Errors are corrected by shifting color from black to white which causes the typewriter to punch the current X, Y coordinates. The carriage and platen are positioned to the error, and the error repeated using a whitening material under the key. Color is then returned to black, causing another set of coordinates to be punched (this set locates the beginning of the erroneous symbols to be deleted), and keying can then resume. This error correction scheme is similar to what is ordinarily used in typing and does not require any tape manipulation or repunching.

INPUT PROCESSING

During Phase I of the system, paper tape input is converted to a standard format by (a) converting input codes to a concise internal computer representation which includes ΔX and ΔY references; and (b) dealing out all coordinates and recording them in a separate table. Input data, as previously described, consists of strings of symbols followed by

an X-Y pointer which refers to the beginning of the symbol string.

This input data structure is altered both by recoding input symbols and by removing X-Y coordinates from the data strings so that X-Y pointers are no longer embedded in the input stream. Symbol recoding is performed by look-up and derives for each input symbol a single character recode as well as a ΔX , ΔY reference. For example, both upper dot (input code 322) and lower dot (input code 77) will have the same symbol recode (1); however, the upper dot will have $\Delta Y = -1$ while the lower dot will carry a $\Delta Y = +1$. A similar distinction will be made (in terms of $\Delta X = 0$ or $\Delta X = 1$) between spacing and non-spacing input characters.

All the input data is read and held in core memory until an end-of-job code is found. The entire graph reference system is then normalized to the left-top corner of the total graph area by adjusting values in the X-Y table to the minimum X-Y values encountered. The X-Y tables are then written out (onto magnetic tape or disk) followed by all the input data which is written in small blocks of 225 symbol- ΔX , ΔY pairs.

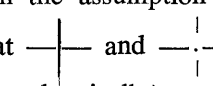
GRID PROCESSING

Phase II of the system re-reads the X-Y table and a single block of recoded input data. The data strings are now to be mapped onto a two-dimensional grid or matrix in which the linear input sequences will be reassembled into a coherent graphic system resembling (in form) the original structure.

The dimensions of the grid are defined dynamically according to the maximum normalized X and Y values of a particular structure. That is, the length of each row is taken to be $X_{MAX} + 2$ and the length of each column is $Y_{MAX} + 4$. (The factors of 2 and 4 beyond the maximum values create an unused border of one column and one row around the structure. This is useful for later processing.) For example, if a structure extends in the X direction a maximum of 22 cells, the 26 cells in the Y direction, grid rows would look like:

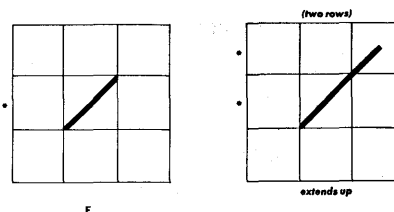
Grid Row	Core Address
(0)	GRID to GRID + 23
(1)	GRID + 24 to GRID + 47
(2)	GRID + 48 to GRID + 71
(3)	GRID + 72 to GRID + 95
:	:
:	:
(26)	GRID + 600 to GRID + 623

The maximum size of the grid is currently defined as 5940 cells or 66 disk sectors.

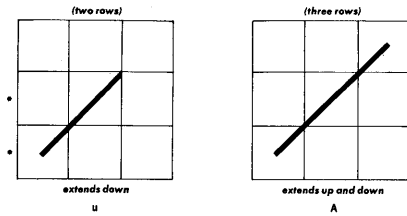
A data symbol is mapped onto the grid by using both the X-Y references for the symbol string and also the ΔX - ΔY factors of the symbol itself. Since the Army Chemical Typewriter uses a three-row range for its input (e.g., /), the ΔY factor (-1, 0, +1) is added to the symbol string Y reference in order to locate the proper grid row (upper, mid, or lower). Similarly, ΔX determines whether or not to step an X index to the next column. When a white color code is encountered, a correct-error flag is turned on. During the correct-error condition, all non-blank input symbols are stored as blanks, and erase the previous contents of the cell into which they are deposited. When a black color code is encountered, the flag is turned off and normal data mapping resumes. If a data symbol is to be deposited in a cell which is already occupied, a check is made. If the two symbols are the same they are treated as one; if the two symbols are different, a vertex symbol (dot) is deposited on the assumption that this is an overstrike and that  are typographically (though not chemically) equivalent.

OUTPUT PROCESSING

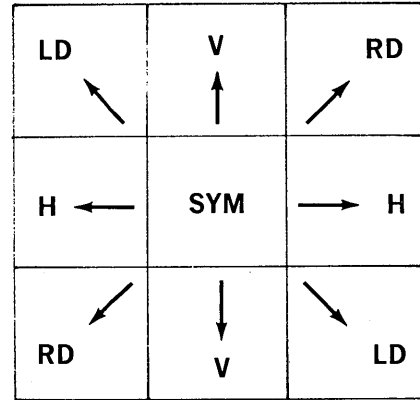
Phase III of the system consists of analyzing the chemical structure as it exists in its grid representation; the analysis is in accordance with rules that will map the grid symbols into Photon control codes. These rules are based largely on the design of the Photon disk. Different chemical structure symbols on the Photon disk (specially designed for the ACS) can extend to 1, 2, or 3 rows of printing. Four variations of this symbol exist on the disk:



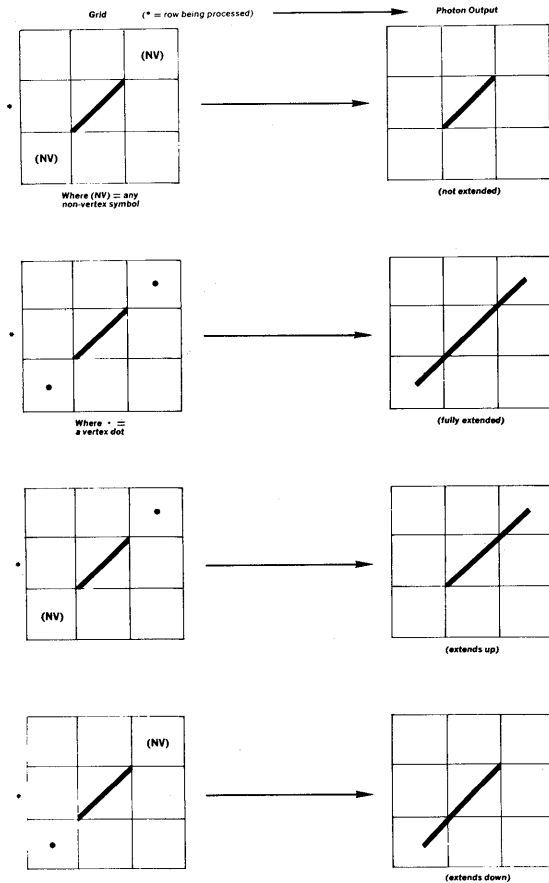
(The single letters represent the Photon disk positions of the symbols.) Each of the four different variations can be used to represent the grid cell symbol /.



The choice among variations depends on what exists in the upper right and lower left cells adjacent to the / cell.



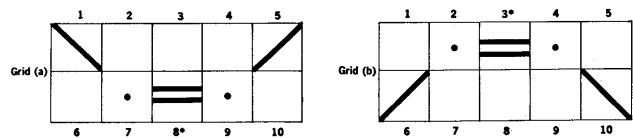
H = Horizontal bond
 V = Vertical bond
 LD = Left Diagonal bond
 LR = Right Diagonal bond



In the case of single bonds, the processing rule is first to consider cells adjacent to the cells being analyzed; each of the four basic single bond symbols has two significant adjoining cells:

A vertex dot discovered in either adjacent cell is a signal to consider a half extended character (2 rows); vertex dots in both adjacent cells indicates a fully extended character (3 rows); no vertex dots in either adjacent cell requires flashing a non-extended character (1 row). If the grid cell being processed (rather than an adjacent cell) contains a vertex, then it is ignored; a grid row consisting entirely of vertices would not produce any output Photon codes except for leading and line drop.

Processing rules for double bonds consist of going through the single bond procedure and then determining whether to place the inner bond (it is a separate disk symbol) on the left or right, or upper or lower side of the single bond. The decision about where to place the inner bond (i.e., which inner bond symbol to use) is made by examining the adjacent bond symbols beyond the adjacent vertices. For example, assume cell 8 is being processed.



The presence of vertex dots in cells 7 and 9 indicates an extended bond is to be used. The presence of “\” in cell 1 and “/” in cell 5 indicates that the inner bond to be flashed is an upper bond. In grid (b) the situation is reversed. Cell 3 is being processed. “/” in cell 6 and “\” in cell 10 means that a lower inner bond symbol is to be selected from the disk.

SUMMARY

Computer programs for the setting of chemical structures, as in the case of our typesetting procedures, are designed to be independent of input or output devices. Our philosophy is to work with available equipment but to design systems capable of utilizing new equipment as it becomes available. For example, in our current work on the setting of structures, we have made use of the several tapepunching typewriters designed to reproduce representations of chemical structures. But already optical scanning for reading chemical structures into an information system has been described by workers at Chemical Abstracts Service.⁸ Character generators have now reached the prototype stage, and these devices with their capability of unlimited character selection and of output speeds upwards of 500 characters per second promise to widen the versatility of our existing systems, both for input and output.

But while the progress of current work and the promise of future hardware developments are speeding the full development of the use of computers in typesetting, the investigation of the role of computerized typesetting has developed an awareness that new concepts of information handling are now possible. The use of computers frees typesetting from its tie with the printed page and opens up a concept of information handling which begins with the encoding process at the time the information is first processed for publication. The ability to handle all

the complexities of scientific material via the computer is an important step toward realizing the full potential of information handling techniques of the future.

ACKNOWLEDGMENT

We wish to acknowledge our appreciation to the National Science Foundation for their support of this work under Grants GN-140 and GN-426.

REFERENCES

1. J. H. Kuney, B. G. Lazorchak, and S. W. Walcavich, *J. Chem. Doc.*, vol. 6, p. 1 (1966).
2. A. Feldman, D. B. Holland, and D. P. Jacobus, *J. Chem. Doc.*, vol. 3, p. 187 (1963).
3. A. Feldman, *Amer. Doc.*, vol. 15, p. 205 (1964).
4. J. M. Mullen, "Atom-by-Atom Typewriter Input for Computerized Storage and Retrieval of Chemical Structures," 150th ACS Meeting, Atlantic City, N.J., Sept. 14, 1965.
5. M. Gordon, private communication, Sept. 1965 (Description of Friden Chemical Typewriter).
6. W. E. Cossam and M. E. Hardenbrook, "Computer Generation of Atom-Bond Connection Tables from Hand-Drawn Chemical Structures," *Proceedings of the American Documentation Institute*, vol. 1, pp. 269-275, Spartan Books, Washington, D.C., 1964.

A COMPUTER-ASSISTED PAGE COMPOSING SYSTEM

Featuring Hyphenless Justification

George Z. Kunkel

*Central Intelligence Agency
Washington, D.C.*

INTRODUCTION

The Central Intelligence Agency has a computer-assisted system for phototypesetting which is producing fully made-up pages of high-quality text composition on positive film. This system was developed by a team of representatives of the Printing Services Division (the Agency's printer) and the Agency's computer people.

The system utilizes both customer tape—prepared as a by-product of manuscript typing—and tape perforated in the composing room. All errors in keyboarding or formatting of data are eliminated before the material is cast on a Photon 513 phototypesetting machine. We make no corrections on the film.

The computer plays an important role in the system, providing interim proofs, performing variable justification arithmetic for the elimination of hyphens, and furnishing a fully formatted Photon control tape. The system accepts a full range of tape, from basic tape furnished by the customer to tape fully formatted by the printer.

We have found that our system makes high-quality type composition truly competitive with typewriter or cold type composition, because of the by-product tape, increased wordage per page, and

because of the ease of keyboarding, editing, and correcting.

The system separates the functions of the author and the printer, allowing both to apply their particular talents to do a job to the best advantage—word composition for the writer, typographic formatting for the printer.

When the Agency was offered the opportunity to present this paper it was asked to describe new or novel features of its computer-assisted phototypesetting system. To me, as a printer, the entire project is new and novel.

First, the development of the system demanded that we examine minutely, existing composing procedures in order to find out what could be changed, and more importantly, what *should* be changed so that we could profitably introduce the digital computer into our typesetting and composing operations. The advent of the computer allowed us to change our procedures drastically and to make real improvements and savings in our composing operations.

Although new equipment has from time to time been available to us, it has never enabled us to change our basic production procedures or workflow. In the past it has always been necessary to perform proofreading and page makeup after typesetting—not because we liked it that way, but because there was no other way. Now, with the aid

of the computer, we can read an intermediate proof, produced on the lineprinter, correct by updating the original file until it is to our liking, and then perform page makeup, receiving a summary page makeup proof of these efforts, and, when everything is deemed proper, request a tape with which a phototypesetting device can cast fully made-up pages on film.

Secondly, because of the intermediate lineprinter proof, we can ask our customers (authors) to supply paper tape along with their manuscript and use this by-product tape for original input to the computer. The printer, by adding to the data via the update capability, can supply format sophistication or whatever else the author was unable to include. Customer tape lessens the keyboarding and proof-reading tasks for the printer and does not penalize the author.

These two advantages just noted are basic to our page composing system. The novelty lies in changed procedures made possible by the computer, and in customer involvement in source machine language.

BASIC FLOW, EQUIPMENT, AND PROGRAMS

The system begins with the production of manuscript by the customer's typist and ends with a fully composed page on film, ready for platemaking.

The major items of equipment and programs used are as follows:

Input devices, producing endless paper tape:

Flexowriter
Dura Mach 10
TTS perforators
Royal McBee typewriter

Processing equipment:

Paper tape to magnetic tape converter
360/65 and peripheral gear
1403 lineprinter
Paper tape punch

Casting equipment:

Photon 513

The three main computer programs used are:

1. *Phase 1 Program*, in which records with indices are created. These data are separated into records of justified lines, each with an index describing the typographic environment of the line as well as the line number and the addresses of words within the line. Phase 1 also handles the updating of Phase 1

records after correction by the printer. Corrections are effected by the punching of paper tape representing the revisions as noted on the lineprinter proof produced by the Phase 1 utility print program. Phase 1 final outputs are a magnetic tape and a lineprinter proof.

2. *Phase 2 Program*, in which the printer's instructions for the arrangement of text on a page are performed. The products of this program are a detail tape for later use with Phase 3, and a summary page makeup lineprinter proof of the results of Phase 2 processing. Phase 2 does not alter the Phase 1 data in any way. It processes data in the Phase 1 tape so as to determine if the page, as formatted by the printer will fit the page image area according to the rules of leading.

3. *Phase 3 Program*, in which the detail tape of Phase 2 and the final updated tape of Phase 1 are processed so as to produce a Photon control tape.

The system flow is this:

<i>Phase 1</i>	<i>Phase 2</i>	<i>Phase 3</i>
Keyboarding	Page makeup	Computer
Computer	Computer	processing
processing	processing	Photon
Proofreading	Verification	casting
Correction, verification		
Computer up-date		

As can be seen from the table, Phase 1 consists of: basic keyboarding (with whatever sophistication the customer is capable of producing); Phase 1 processing (justification and record establishment); proofreading and copy preparation; correction keyboarding and verification; and Phase 1 reprocessing for updating of the file.

Phase 2 consists of: page makeup instructions (keyboarded by the compositor); Phase 2 processing; and verification of a summary print.

Phase 3 consists of: Phase 3 processing to produce Photon control tape; and casting of material on the Photon.

With this basic flow in mind I will now run through it in some detail and emphasize the things which we feel are important.

DETAILED FLOW, PROCEDURES, AND PROGRAM DESCRIPTIONS

Basic keyboarding has received considerable effort from our group. We have developed typewriters with considerable editing capabilities. These

machines, in the hands of competent typists, can greatly simplify and expedite redrafting of revised manuscript in the customer's office.

The typist can, while revising a tape, take controlled jumps at 40 characters per second by word, line, paragraph, or page. Insertions, deletions, and substitutions to the original tape can be made swiftly and accurately.

Generally speaking customer tape contains very little sophistication insofar as typographic formatting is concerned.

The reasons for this are that writers or authors are primarily concerned with the production of their writing (basic data) and not the ultimate format, and typists, for the most part, are not knowledgeable of typographic requirements.

After the basic typing is completed the customer forwards his tape for computer processing. Then there are two elective procedures: (1) the customer can update the lineprinter proof, or (2) he can forward it to the printer for formatting and production.

Phase 1 processing is accomplished as follows: The paper tape is transferred to magnetic tape and the data are then fed into the Phase 1 program. The text is justified according to the font, point size, and measure prescribed in a control message which precedes the data. Each justified line or record is accompanied by a program-generated index which describes all the typographic features, or environment of the line as well as its record number and the core position of each word in the line.

This is the method by which the "endless" data are broken up into records and the record index is established: The amount of alphanumeric data to be contained on a record is determined by the rules for justification. The variable justification technique makes this step swift with no detours for hyphenation routines. During the justification step the typographic facts of the line are recorded in an index. Also recorded in the index are the addresses of the first letter of each word in the line. This is extremely useful to the update routine. The total facts recorded are:

- Line number of record.
- Quad condition of line. This tells how the deficit space should be distributed in the line.
- Lens size.
- Measure.

Set-size and set-multiplier. These are data necessary for Photon casting.

Font condition of first character, and Shift condition of first character. These two items are essential when updating. Presence of accents, symbols, or piece fractions. These data are used in the print and punch programs.

Deficit space to be generated for justification. This is used in punch program. Line-ending code. Necessary to the update. Error indicator. This is used by the print program.

Extract flag.

Update indicator. Used to flag lines on an updated lineprinter proof.

Normal vertical value of line. Used by the page makeup program and print program.

Expansion and contraction factor for vertical space. Used by the page makeup program.

Total paragraph vertical space. This appears as a paragraph total on the lineprinter proof.

Column vertical value. Used in page makeup.

Special indention indicator. Used by the punch program and print program for special formats.

Word number index. Used by the update routine.

Ultimately the record length, after Phase 1 processing, becomes 504 positions, 300 for data, and 204 for the index. The index is utilized on subsequent passes to supply information necessary to the update routine, the print program, the page makeup program, and the punch program. The program was written originally in Autocoder for the 7010 and amounted to 36 inches of tab cards. It has subsequently been rewritten for the 360 in ALC (assembly language coding).

The justification processing is fast—in the neighborhood of 275 records per minute for original Phase 1 processing. Updating speed depends on the number of corrections. Only revised words and lines need to be run through Phase 1 justification. Unchanged material is transferred intact to the new Phase 1 output tape.

Phase 1 will process a tape with any degree of

typographic complexity which a typist is capable of entering into the record. However, as mentioned previously, most of our customers are not interested in producing more than basic data tape, and are relying on the printer to apply format indicators and other refinements by way of the Phase 1 update. The Phase 1 program will not discontinue because of input errors. This can be stated as a "Garbage in, Goodies out" program. Keyboarding is very simple—it consists of a basic font and, with the use of the start and stop precedence codes the entire capability of the composing device is available at the end of the system. There are 16 floating accents, 360 symbols, font selection (12 fonts in 12 point sizes), as well as many other things. All are easily keyed by an operator.

Proofreading and copy preparation are the primary tasks performed at this stage. This is not the traditional paired reading, it is called a "silent reading." The purpose of this reading is to read for sense, mark typographic errors, indicate special symbols and accents, and to perform typographic formatting. Paired reading has been eliminated for that material produced from customer tape.

The system provides the reader with a lineprinter proof which is as easy on the eyes as we could make it. Capitals and lowercase letters, superiors for word numbering, floating accents (Fig. 1).

The proof is informative—containing all the appropriate typographic information needed by a proofreader in a clean and unambiguous manner. Notice the accents, format indicators, vertical values for paragraphs. This proof gained immediate acceptance by our proofreaders because of ease of reading and because it presented the necessary information to them. It is an incomparable improvement on the all cap proof usually available from a lineprinter.

After proofreading, corrections are punched on a Flexowriter—all calls for typographic formatting, accents, symbols, and special formats appear in red (second line of Fig. 2), as the ribbon shift and ribbon unshift are the precedence start and stop codes, respectively.

This hard copy serves as the revised proof and is verified by a proofreader for accuracy before the tape is forwarded for processing.

The eight digits on the left are the line and word addresses of the beginning and ending of the correction address. A correction can only affect the area addressed and the Phase 1 program is written to

maintain the font and shift integrity of material without the confines of the correction address. The maintenance of integrity by the computer eliminates any possibility of inadvertently damaging text elements other than those addressed. This feature of the program, we feel, is valuable and unusual.

The correction data are then run through Phase 1 against the original record, and again a proof is produced. This proof is then used to perform page makeup instructions.

Page makeup instructions are prepared by the printer after the Phase 1 proof has been declared correct. The human processes for this stage are makeup planning and keyboarding of the instructions on a Flexowriter. Makeup planning is done with the use of the Phase 1 proof, an adding machine, and a planning form on which the makeup planner encribes his decisions. The first action is to prepare the page makeup program control message. This message, which requires 72 keystrokes, contains the basic parameters for processing. Information included in this message is:

- Program selection indicators.
- Job number.
- Number of columns.
- Beginning page number.
- Whether running heads are to be generated.
- Whether running feet are to be generated.
- Whether out-of-reading-sequence material (inserts) will be used.
- Space between columns.
- Horizontal image width.
- Space between running head and text.
- Space between text and running foot.
- Basic column width.
- Allowable expansion and contraction limits for leading.
- Vertical depth of the page.

The second step to page makeup is to determine the Phase 1 lines to be included on a page. The Phase 1 proof displays the normal leading value of each paragraph. The value is expressed in points (1/72 of an inch). The makeup planner knows that each Phase 1 record's vertical value may be expanded or contracted as instructed in the control message. His problem lies in prescribing a suitable number of Phase 1 records to a page which by program-controlled leading expansion or contrac-


```

001 76      old Mr. Rodman died, in the fall of 1790; and, in the
01 02 03      04      05 06 07      08 09      10      11 12
001 77      ensuing winter, both his daughters perished of the small-
01      02      03      04      05      06      07 08 09
001 78      pox, within a few weeks of each other.*Shortly afterwards
01 02      0304 05      06 07      08      09      10
001 79      (in the spring of 1791), Mr. Julius Rodman, the son, set
01 02 03      04 05      06 07      08      09 10      11
001 80      out upon the expedition which forms the subject of the
01 02 03 04      05      06      07 08      09 10
001 81      following pages.*Returning from this in 1794, as
01      02      03      04      05      06 07      08
001 82      hereinafter stated, he took up his abode near Abingdon,
01      02      03 04      05 06 07      08      09
001 83      in Virginia, where he married, and had three children,
01 02      03      04 05      06 07 08      09
001 84 Q1    and where most of his descendants now live.
01 02      03      04 05 06      07 08
001 85      *We are informed by Mr. James Rodman that his
01 02 03      04 05 06      07      08 09
001 86      grandfather had merely kept an outline diary of his tour,
01      02 03      04      05 06      07      08 09 10
001 87      during the many difficulties of its progress; and that the
01      02 03      04      05 06 07      08 09 10
001 88      MSS. with which we have been furnished were not
01      02 03      04 05      06 07      08 09
001 89      written out in detail, from that diary, until many years
01      02 03 04      05 06 07      08      09 10
001 90      afterwards, when the tourist was induced to undertake
01      02      03 04      05 06      07 08
001 91      the task, at the instigation of M. André Michau, the
01 02      03 04 05      06 07 08      09      10
001 92      botanist, and author of the >f02<Flora Boreali-Americana,
01      02 03      04 05 06      07      08
001 93      >f01<and of the >f02<Histoire des Chenes d'Amérique.*>f01<M.
01      02 03 04      05 06      07      08
001 94      Michau, it will be remembered, had made an offer of his
01      02 03      04 05      06 07      08 09      10 11
001 95      services to Mr. Jefferson, when that statesman first
01      02 03 04      05      06 07      08

```

Figure 1. Phase 1 lineprinter proof.

tion, will suffice to produce a vertically justified column. The lines to be included are designated by the inclusive line addresses as shown on the Phase 1 proofs, and a column prefix letter (leftmost column is "a"). As an example the instruction for a one-column page might be "a00107-00157." This instruction obviously, would result in a final page with the 51 lines addressed contained in it. The lines will have been adjusted (leaded) to comply to the prescribed depth by the Phase 2 program. Another tool of page makeup consists of an instruction to insert material from a Phase 1 stepchild tape containing material for footnotes or figure captions. This material was identified at Phase 1

keyboarding time by an extract request e5. The Phase 2 program searches the Phase 1 records and transfers all lines so flagged to an insert tape. This material is requested by the makeup planner by adding an additional prefix letter "i" to the line address required. For example, "ai00107-00157, ai00101-00106.", would result in a one-column page with a footnote.

The makeup planner must also provide space within columns for the insertion of graphics. This job cannot be done prior to page makeup, for the exact position of the figures cannot be determined without simultaneously knowing the pagination breaks. A figure insertion is requested as "af270,"

```

J. 1-20815 Update 7-21-66
c1,r,1,1-20815,101-101,f01-m332-110-v00720,08-15-66,6,
p001
14061407giantic
31083109think
p002
93019302Greenwich
p003
23022803forty-
zend
End Update

```

Figure 2. Flexowriter proof (lines 2 and 10 are in red in the original).

which means that column "a" will have 270 points of blank space reserved for a graphic insert. For example, "a00107-00137,ai00101-00106,af270", would result in a page with 31 lines of text and the blank space for a picture. Vertical space requests may be assigned to one, two, or three columns by prefixing, as "abc270". This would place blank space of 270 points in depth in three columns. Text lines may also be multicolumn-assigned in the same manner.

Subsectioning of pages, that is a specific vertical depth justification within the page depth justification may also be accomplished. In this case the planner describes the depth of the subsection and then the elements to be equalized, as: "s250,a00101-00120xb00121-00140;" which requests that the "a" column and "b" column material each be made equal to 250 points in vertical depth.

The sequence of elements within the instruction determines their ultimate sequence on a page.

Keyboarding consists of punching a tape from the makeup planner's form. This is done on a Flexowriter. The hard copy is verified before the tape is forwarded to the computer room (Fig. 3).

The Phase 2 program performs 3 major tasks: column segregation, leading contraction and/or expansion, and the creation of a detail tape from which a Summary Page Makeup Proof can be generated, and which can also serve as input to Phase 3. Input consists of:

- The ultimate Phase 1 tape.
- The page makeup instruction tape.
- The Phase 2 program tape.

Output consists of:

- A detail tape.
- A summary listing (for verification by the printer).

This page makeup system bypasses entirely the traditional metal assembly steps and equipment.

The summary listing flags any page makeup instruction which has not met the specifications for leading contraction or expansion to accomplish columnar justification (Fig. 4). Notice also that the running heads and feet have been generated for each page.

The Phase 3 program utilizes the output of the Phase 2 program and the Phase 1 tape to construct an output tape which will control the off-line punching of the Photon 513 control paper tape. The program arranges the lines for sweep casting. It calculates the leading codes to be output—these are not the same as those indicated in the column tapes—they represent the comparative difference in vertical location of column lines cast in a sweep. The program also generates running heads and feet and page numbers. The program interprets all typographic instructions and translates them to Photon machine language.

Flexibility is a feature of the system—it will accept unverified 6, 7, or 8 level tape—typography is limited only by the capability of the casting device. The material can be updated as often as necessary. Page makeup can be 1, 2, or 3 columns, with or without illustrations.

PROGRAM SUMMARIES

Now a word about the computer programs used with this system. They are quite complex. They bear

```

J. 1-30815 Page Makeup 7-28-66
c2,p,1,1-30815 ,101-101,e3,v00720,g025,p001,h018,f018,y,y,y,
e2,c1,w216,
a00116-00177,a100107-00110,s384,b00178-00212xc00213-00246,
c100111-00111;b0100112-00114,b00380-00404,c00405-00428,
c100115-00115.
zend
End Page Makeup

```

Figure 3. Page makeup instruction proof (lines 2, 3, and 7 are in red in the original).

JOB NO. 1-20815

VERT.DEPTH 00720 NO. OF COL. 2

F11

THE JOURNAL OF JULIUS RODMAN

SECTION

COL.

ELEMENT

A

00229-00289.

B

00290-00349.

F11 VOL IV

014

PAGE 002

Figure 4. Summary page makeup proof.

very little relationship to computer programs written for a slug-casting machine operation. The specifications for the Phase 1 program include an updating requirement. The instructions for this requirement were that it must be "as conservative as possible of keystrokes to an operator." Consequently eight keystrokes are all that are required for addressing the beginning and end point of a correction. A correction may be the deletion of data, addition of data, or the substitution of new data for old.

The Phase 1 program must deal with 17 interacting factors which can modify a line. A change in any of these factors always affects the bookkeeping (addresses) and typographic environment of a line, and consequently the line index. The program must remember the shift and font modifiers of every word in a line. The update portion of the program has been written so as to maintain the typographic characteristics of material not addressed for correction. Thus a keyboard operator need only concern himself with correcting errors and not confuse himself with concern for the effect of his corrections on the data subsequent to, or following the correction address. We have dubbed this feature "maintenance of typographic integrity."

The Phase 1 program accepts unverified input data—the only thing that must be flawless is a 50-character control message to initiate the program

and to set up the constants necessary for the program to begin processing.

The program handles 52 discrete input codes and considerable precedence-code-bracketed material by which typographic requirements are related to the text. The program handles measure change, font change, lens change, quadding, indexing, floating accents, special symbols, and piece fractions as well as compound instructions for special formats within a job.

Keep in mind that the program handles these things in the update routine as well as when originally processing.

Input errors in the precedence-code-bracketed material are flagged by the computer and the questionable data printed in the text. The program continues processing, using the last known parameters. The ease and flexibility of updating makes correction of these flaws in data very easy for the operator.

The print program for Phase 1 also does a yeoman job. Besides printing capital and lowercase letters it produces any of 16 floating accents over or under (whichever is appropriate) any letter on the chain.

The page makeup program—Phase 2—adjusts leading values in from one to three columns of text, justifies the columns to be a specified overall depth

and will, when instructed, justify portions of columns within the overall depth to prescribed parameters.

It produces data for running heads and running feet, controls page numbering, inserts footnotes and vertical space for graphics, and then produces a summary proof of the pages it has "theoretically" constructed, with occasional chiding remarks to the page makeup man when he has asked for some impossible page design.

Phase 3 collates the records of a Phase 1 tape into their column sequences as indicated by the Phase 2 detail tape and then reconstructs them in casting sequence for the Photon. It must prescribe the most efficient XY movements to accomplish casting and punch all characters and functional control codes for the Photon.

The programs represent the product of a magnificent effort by the data processing people of the Agency to write programs that are useful, dependable, and efficient.

HYPHENLESS JUSTIFICATION *

An important and, we believe, novel feature of our system is hyphenless justification. By taking advantage of the Photon's set-size-change capability, the Phase 1 program seldom has to produce a word-dividing hyphen. This results in extremely rapid processing in Phase 1 and allows the use of core and peripheral gear for things other than hyphenation logic. Set size variation is NOT letterspacing. It produces words whose horizontal letter proportions are not changed as they are in letterspacing. This is of benefit to the reader because it is visually smoother and because end-of-line hyphens do not interrupt his reading.

The arithmetic advantage can be explained as follows:

Metal linecasting justification is accomplished by adding the widths of the characters in a line to the widths of the expanded interword spaces so as to achieve a predetermined measure.

$$\text{width of characters} + \text{spaces} = \text{line length}$$

This arithmetic allows for one variable—the spaceband. Using this system you will get a certain number of end-of-line hyphens for a given number of

lines, depending on the type size, measure, and alphabet length of the font.

The standard justification arithmetic for the Photon is approximately the same:

$$\begin{aligned} &\text{width of characters} + \text{spaces} \times \text{set size} \\ &= \text{line length (in units)} \times \text{set size} \end{aligned}$$

Here is the arithmetic as before in the linecasting justification with still one variable—the interword space. There is one other factor in the formula, the set size. This must be in the computation in order to allow for the horizontal area of the letters at the different point sizes available. Twelve different point sizes can be set from one font. Still the same result—hyphens.

The following formula shows a variation in the arithmetic that gives us hyphenless justification:

$$\begin{array}{ccccccc} \text{width of} & & \text{set size 1} & & & & \\ \text{characters,} & & \text{or} & & & & \\ \text{in units} & & \text{set size 2} & & \text{line} & & \\ + & \times & \text{or} & = & \text{length,} & \times & \\ \text{spaces,} & & \text{set size 3} & & \text{in units} & & \text{set} \\ \text{in units} & & \text{or} & & & & \text{size} \\ & & \text{set size 4} & & & & \end{array}$$

It works like this: Make two of the factors variable; make the set size variable as well as the interword space.

Figure 5 shows the effective increase in justification range by comparing the standard method for justification to our method. The band at the top of the figure represents the units of relative value necessary to set a 20-pica line using one set size—10. It points out that the justification range of this line, assuming it had 5 spaces in it, would be 50 relative value units. The minimum for a space being 4 units and the maximum desirable space 14 units or 10 units of expansion for each space. The computer must turn out a line with precisely 426 units in it, and find a place to end the line within the justification range of 50 units—this is approximately 7 characters. If it can't find a legitimate line ending point it has to try to find a place to drop in a hyphen and it has to find the right place.

The second band (Fig. 5) represents the same measure but shows the increased range accomplished by allowing a choice of set sizes—in this case four. The 4 set sizes give us a choice of 4 lines measured in terms of the relative value to accomplish our 20 pica measure—from the minimum of 387, to the maximum of 448, a range of 61 units. This, added to the range of the 5 spaces gives us a total range

* Hyphenless justification was the subject of articles which appeared in the April 1965 issues of *Datamation* and *Printing Production*.

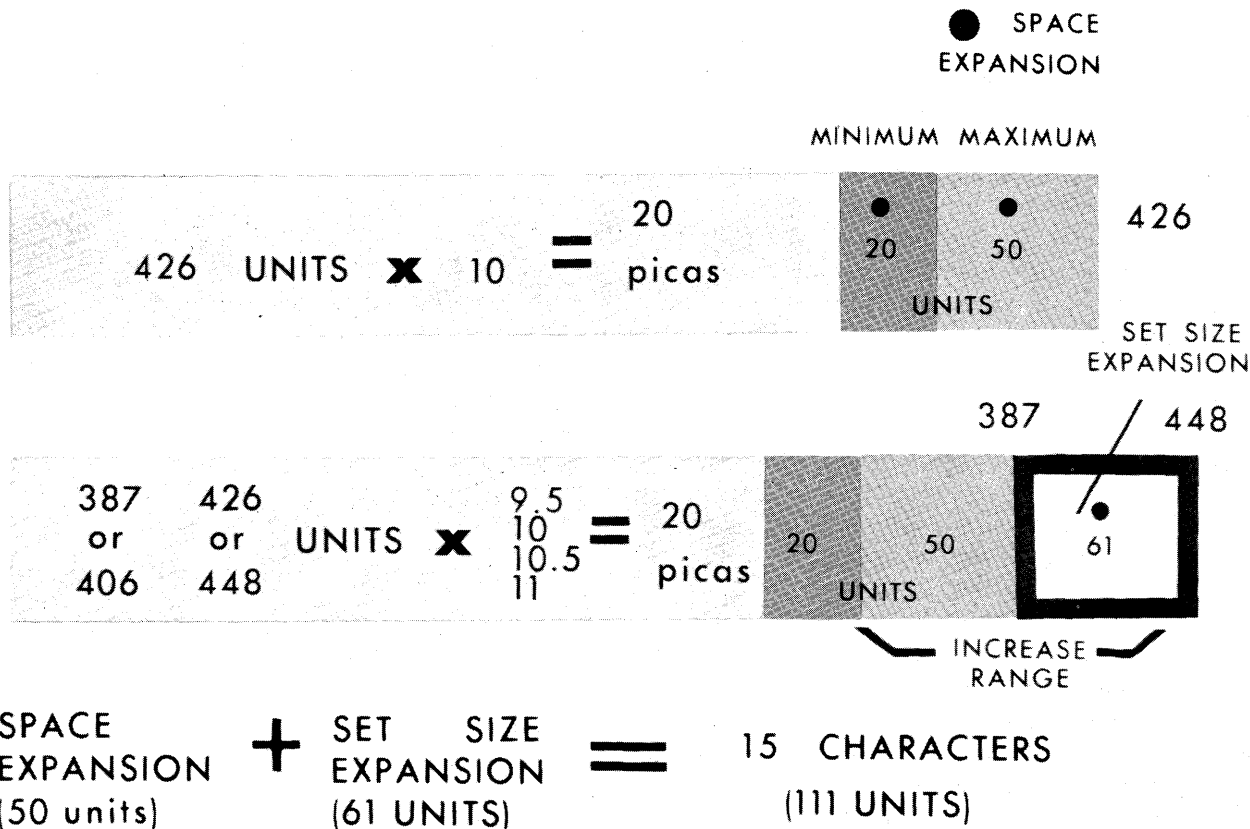


Figure 5. Justification range comparison for a 20 pica line with 5 spaces.

of 111 units or 15 characters. This is quite an edge to carry into the fray. In order to stall this system a 16-character word would have to be the last word in the line. From experience, we have determined that this system will go indefinitely without the need for an end-of-line hyphen. We are using it primarily on a 20-pica line of 10-point type and very rarely have had to put in a hyphen. As in normal typesetting the fewer the spaces, the shorter the justification range, but as long as we have a single interword

space at 20 picas we have 71 units in which to get an end-of-line hit. When the computer does get stalled on a rare line, it is programmed to drop the hyphen in the offending word using the normal set-size value, so that there will be plenty of room for a printer to make a correction on a subsequent pass through the computer. Most of the lines are produced at the normal (that is the recommended) set size. The extremes occur only occasionally.

Figure 6 shows hyphenless text.

unexplored region." It is, moreover, the only unexplored region within the limits of the continent of North America. Such being the case, our friends will know how to pardon us for the slight amount of unctious with which we have urged this Journal upon the public attention. For our own parts, we have found, in its perusal, a degree and a species of interest such as no similar narrative ever inspired. Nor do we think that our relation to these papers, as the channel through which they will be first made known, has had more than a

1784 (being then about eighteen years of age), with his father and two maiden sisters. The family first settled in New York; but afterwards made their way to Kentucky, and established themselves, almost in hermit fashion, on the banks of the Mississippi near where Mills' Point now makes into the river. Here old Mr. Rodman died, in the fall of 1790; and, in the ensuing winter, both his daughters perished of the small-pox, within a few weeks of each other. Shortly afterwards (in the spring of 1791), Mr. Julius Rodman, the son, set out upon the

contemplated sending an expedition across the Rocky Mountains. He was engaged to prosecute the journey, and had even proceeded on his way as far as Kentucky, when he was overtaken by an order from the French Minister, then at Philadelphia, requiring him to relinquish the design, and to pursue elsewhere the botanical inquiries on which he was employed by his government. The contemplated undertaking then fell into the hands of Messieurs Lewis and Clarke, by whom it was successfully accomplished.

Continued on Page 48

Figure 6. An example of hyphenless text.

This feature of the system is indeed unique, both for the fact that for all practical purposes we have no hyphens, except for compounding, and because of the method by which end-of-line hyphens are avoided. We have perhaps the only computerized typesetting system which produces hyphenless composition of high quality without letterspacing. This is why we have not had to develop a hyphenation program. We can use our computer capacity for more useful purposes.

MISCELLANY

Hyphens

Some of the thinking and sweating that went into our decision to go with this system of hyphenless justification went like this:

The hyphen at the end of a line is a device invented years ago to solve a problem—it does not help a reader or improve understanding, or serve any grammatic purpose. Its use is indefensible except as a method of ending a justified line when the justification method cannot end the line at a natural breaking point—the end of a word.

Other niceties of typography are necessary and defensible—capital and lowercase letters, justification, variable width letters—and at the risk of sounding pedantic, I think it is important to go on record as to why these niceties are necessary.

Upper and Lowercase Print Chain

In the development of goals, and subsequently, the specifications for the computerized typesetting system and the concomitant computer programs, the computer members of the team properly asked the printers about the necessities of our time-honored requirements of typography relating to the composition of text.

These questions sound childish but what they point out is that in the marriage of composing and computing a *modus vivendi* must be worked out which is based on real need and real capability, not on maintaining habits and traditions whose reason for existence is lost in antiquity.

The first question was: "Why do you need capital and lowercase letters on your print chain?" These facts were offered:

Capitals and lowercase letters are more "legible"—this means "capable of being read or deciphered."

Lowercase letters are less subject to misreading, as for example compare the capitals Q and O, or F and E to the minuscules q and o, and f and e.

Groups of letters comprising a word produce a more recognizable shape when composed of minuscules rather than majuscules. Words produced in all caps produce only quadrangular shapes varying only in length. Lowercase words not only vary in length but the presence of letters with ascenders and descenders creates distinctive word shapes.

The use of both capital and lowercase letters is essential to assure that proper interpretation of the thoughts of a writer are passed on to the reader without ambiguities. For example, this question in all caps or all lowercase defies accurate interpretation:

WHERE IS JOHN BROWN?

The question could mean: Where is John brown?
or Where is John Brown?

Monospacing

A question related to the previous one is: Why do you want letters of variable width? Why not monospacing?

The shape and size of the letters of the alphabet have evolved by what can be compared to the "law of natural selection."

The letters we use today are the survivors of thousands of symbols which have in the past been used to represent sounds. Each letter of our present alphabet is separately recognizable by its design and comparative shape as well as being capable of melding with adjacent letters to form words which are a distinct entity even though comprised of singularly discrete symbols.

Monospacing requires that letters which should be *naturally* slim be made wider and conversely letters *naturally* wide be compressed so as to conform to a monospacing norm. This may be mechanically advantageous but the advantage is gained at the expense of the reader. The crudity of the printed material is accepted by the reader as a necessary evil but not as desirable.

Even Right Margin

The next question the computer people asked us was: Why do you need to justify your text lines? Why not let them run ragged?

To defend this requirement I will offer admittedly, a somewhat abstruse answer, replete with logic, perhaps not Boolean, but still logic.

A line which does not fill a measure marks the end of a paragraph.

Words are elements of a sentence, a sentence is a complete thought, and a paragraph is a group of related thoughts. Therefore the existence of a justified line implies to the reader that the group of related thoughts has not ended and that more will follow. Conversely the short line indicates the end of a group. Whether or not the reader is consciously aware of the reasons for justification is not important—what matters is that his mind is led effortlessly and naturally from thought to thought and group to group. And, of course, there can be no question but that even right-hand margins greatly improve the appearance of a page.

Agreement of all team members to the necessity for capital and lowercase letters, variable width characters, and justification of lines of text was attained. The problem that next presented itself was determining how to chop up a fixed sequence string of characters so as to fill a line of a prescribed measure.

More on Hyphens

At first, we investigated existing information on programmed hyphenation. We quickly came to the conclusion that there was no foolproof way to accomplish proper hyphenation with complete accu-

racy, and that whatever route we took for doing it—logic, dictionary, or a combination of the two, the program would be large and use up a sizable amount of computer core and processing time.

The use of fixed letterspacing was discarded as a means of extending the justification range because it destroyed the proportional balance of the letters. Increasing the acceptable interword space was also discarded as being disruptive to the reader. When we hit upon varying the set size to increase justification range it enabled us to preserve letter balance and control interword space. Variable set-size justification solved our problem—without sacrificing readability.

SUMMARY

In summary, we feel that a few of the novel features of our system are:

1. Customer tape—tape produced as a by-product of manuscript typing.
2. The change in workflow made possible by the computer.
3. The cap and lowercase lineprinter proof with floating accents.
4. The environmental index to Phase 1 records.
5. Maintenance of integrity of shift, font, and measure during updating.
6. Hyphenless justification.
7. Page makeup flexibility.

A GENERAL METHOD FOR PRODUCING RANDOM VARIABLES IN A COMPUTER

George Marsaglia

*Boeing Scientific Research Laboratories
Seattle, Washington*

SUMMARY

Many random variables can be approximated quite closely by $c(M + U_1 + U_2 + U_3)$, where c is constant, M is a discrete random variable, and the U 's are uniform random variables. Such a representation appears attractive as a method for generating variates in a computer, since $M + U_1 + U_2 + U_3$ can be quickly and simply generated. A typical application of this idea will have M taking from 4 to 7 values; the required X will be produced in the form $c(M + U_1 + U_2 + U_3)$ perhaps 95–99% of the time, and occasionally by the rejection technique, to make the resulting distribution come out right. This paper describes the method and gives examples of how to generate beta, normal, and chi-square variates.

INTRODUCTION

Given a sequence of independent uniform random variables U_1, U_2, \dots , we are concerned with methods for representing an arbitrary variate X as a function of the U 's. Such representations lead to programs for generating X 's in a computer. There are any number of ways to represent X as a function of the U 's; the problem is to find those which lead to fast, accurate, easy to code programs which occupy little space in the computer.

Programs which are very fast and accurate may be

based on representing the density of X as a mixture of "easy" and "difficult" densities, the latter being called for infrequently. See Ref. 2 for a general discussion and Refs. 1 and 3 for applications to normal and exponential variables. These programs are very fast, but at the same time they are rather complicated and require hundreds of stored constants. In this paper, we will try to develop a general method that is simpler, but still very fast. The procedure will be explained by way of three examples (a beta variate, the normal distribution, and a chi-square variate), from which the suitability of the method and its general applicability can be inferred. These points are discussed in the final section.

Throughout this article, we will use $f(x)$ to represent the density of $U_1 + U_2 + U_3$, with the U 's independent, uniform over $[0,1]$. This density is a piecewise parabola, once differentiable, or, if you prefer, a parabolic spline curve:

$$f(x) = \begin{cases} .5x^2 & 0 \leq x \leq 1 \\ .5x^2 - 1.5(x-1)^2 & 1 \leq x \leq 2 \\ .5x^2 - 1.5(x-1)^2 + 1.5(x-2)^2 & 2 \leq x \leq 3 \\ 0 & x < 0 \text{ or } x > 3. \end{cases}$$

A BETA VARIATE

We want a method for generating a variate X with a beta density, say $b(x) = 105x^4(1-x)^2$, $0 < x < 1$. Our aim is to generate X in the form $c(M + U_1 +$

$U_2 + U_3$) most of the time, occasionally generating X by the rejection technique in order that the resulting mixture be correct. A good choice for c in this case is .1. We will generate X in the form $.1(M + U_1 + U_2 + U_3)$ with as high a frequency as possible. That is, we will put $X = .1(0 + U_1 + U_2 + U_3)$ with probability p_0 , put $X = .1(1 + U_1 + U_2 + U_3)$ with probability p_1 , put $X = .1(2 + U_1 + \dots + U_3)$ with probability p_2, \dots , and put $X = .1(7 + U_1 + U_2 + U_3)$ with probability p_7 . It turns out that we must put $p_0 = 0$, since $b(x)$ varies as x^4 , while $f(x)$ varies as x^2 , at the origin. We can, however, fit $b(x)$ very closely with a mixture of the densities of $.1(1 + U_1 + U_2 + U_3), .1(2 + U_1 + U_2 + U_3), \dots, .1(7 + U_1 + U_2 + U_3)$. We may formulate the problem of finding the best set of p 's as follows:

Choose p_1, p_2, \dots, p_7 so as to maximize
 $p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7$

subject to the condition that $p_i \geq 0$ and

$$p_1[10f(10x-1)] + p_2[10f(10x-2)] + \dots + p_7 [10f(10x-7)] \leq 105x^4(1-x)^2 \text{ for } 0 \leq x \leq 1 \quad (1)$$

This optimization problem is similar to those of linear programming—in fact, if we specify condition (1) for a suitably fine mesh of x values, we have an ordinary linear programming problem. We find that we can get $p_1 + \dots + p_7$ very close to 1, in fact $\sum p_i = .9915$, and still maintain condition (1). Thus we write

$$105x^4(1-x)^2 = \sum_{i=1}^7 p_i[10f(10x-i)] + .0085h(x) \text{ for } 0 \leq x \leq 1,$$

where the p 's are, in order, .0199, .0633, .1297,

.1978, .2369, .2155, and .1284, and $\sum p_i = .9915$. The residual function $g(x) = .0085h(x)$ is drawn in Fig. 1. We may generate X with density $h(x)$ by the rejection technique. We summarize with this outline:

To generate a beta variate X , density $105x^4(1-x)^2$, $0 < x < 1$,

1. with probability $p_5 = .2369$ put $X = .1[5 + U_1 + U_2 + U_3]$
2. with probability $p_6 = .2155$ put $X = .1[6 + U_1 + U_2 + U_3]$
3. with probability $p_4 = .1978$ put $X = .1[4 + U_1 + U_2 + U_3]$
4. with probability $p_3 = .1297$ put $X = .1[3 + U_1 + U_2 + U_3]$
5. with probability $p_7 = .1284$ put $X = .1[7 + U_1 + U_2 + U_3]$
6. with probability $p_2 = .0633$ put $X = .1[2 + U_1 + U_2 + U_3]$
7. with probability $p_1 = .0199$ put $X = .1[1 + U_1 + U_2 + U_3]$
8. with probability .0085 generate X with density $h(x)$ by the rejection technique outlined in Fig. 1.

Referring to Fig. 1, we see that the residual function $g(x) = b(x) - \sum p_i[10f(10x-i)]$ has a peak on the right end which makes the rejection technique too inefficient; this difficulty may be overcome in several ways—for example, by adding one more step to the procedure, as described in the figure. This will add another step to the outline:

7a. with probability .0044, put $X = .05(17 + U_1 + U_2 + U_3)$,

and the probability in step 8 will be changed from .0085 to .0041.

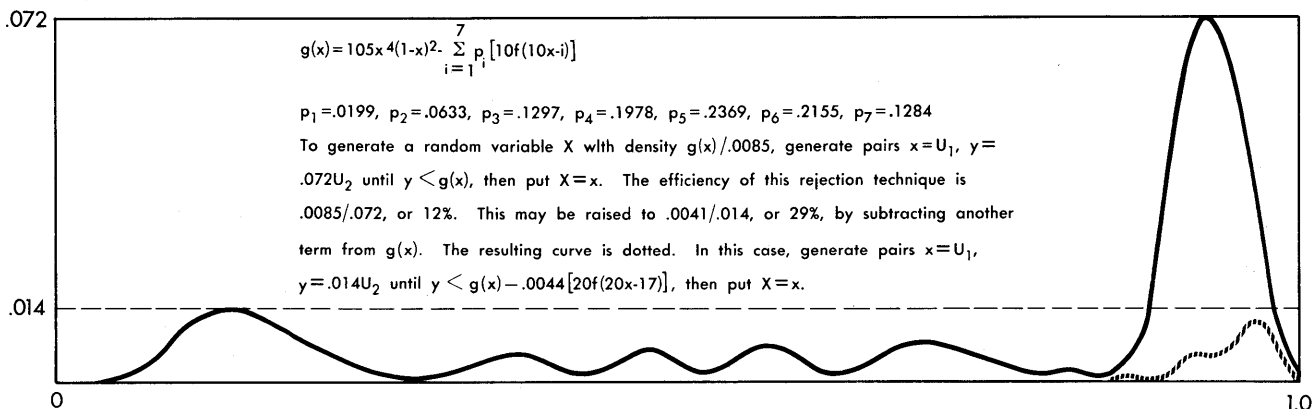


Figure 1. Method for generating a variate from the residual portion of the beta distribution, by the rejection technique.

THE NORMAL DISTRIBUTION

We want a method for generating a standard normal variate X , density $(2\pi)^{-.5}e^{-.5x^2}$. We temporarily discard the tails, $|x| > 3.5$, and divide the interval $-3.5 < x < 3.5$ into 10 parts. We will generate X in the form $.7(M + U_1 + U_2 + U_3)$, where M takes values $-5, -4, -3, -2, -1, 0, 1, 2$ with probabilities p_1, p_2, \dots, p_8 . We choose the p 's so as to maximize the frequency of the representation $X = .7(M + U_1 + U_2 + U_3)$, as follows:

Choose p_1, p_2, \dots, p_8 so as to maximize

$$p_1 + p_2 + \dots + p_8$$

subject to the condition that $p_i > 0$ and

$$p_1 \left[\frac{10}{7} f\left(\frac{10}{7}x+5\right) \right] + p_2 \left[\frac{10}{7} f\left(\frac{10}{7}x+4\right) \right] + \dots + p_8 \left[\frac{10}{7} f\left(\frac{10}{7}x-2\right) \right] \leq (2\pi)^{-.5} e^{-.5x^2}$$

for $-3.5 \leq x \leq 3.5$ (2)

The solution is $p_1 = p_8 = .0092$, $p_2 = p_7 = .0517$, $p_3 = p_6 = .1576$, and $p_4 = p_5 = .2767$, with $p_1 + \dots + p_8 = .9904$. Thus we write

$$(2\pi)^{-.5} e^{-.5x^2} = \sum_{i=1}^8 p_i \left[\frac{10}{7} f\left(\frac{10}{7}x-i+6\right) \right] + .0091347418h(x) + .0004652582t(x)$$

where $h(x)$ is the residual density on $-3.5 \leq x \leq 3.5$, and $t(x)$ is the tail, i.e., the density of X , conditioned by $|x| > 3.5$. We generate X with density h by the rejection technique (Fig. 2), and from the tail by the method described in Ref. 4. These are steps 9 and 10 in the following outline:

To generate a standard normal variate X , density $(2\pi)^{-.5}e^{-.5x^2}$,

1. with probability .2767, put $X = .7(U_1 + U_2 + U_3 - 1)$
2. with probability .2767, put $X = .7(U_1 + U_2 + U_3 - 2)$
3. with probability .1576, put $X = .7(U_1 + U_2 + U_3 - 0)$
4. with probability .1576, put $X = .7(U_1 + U_2 + U_3 - 3)$
5. with probability .0517, put $X = .7(U_1 + U_2 + U_3 - 4)$
6. with probability .0517, put $X = .7(U_1 + U_2 + U_3 + 1)$
7. with probability .0092, put $X = .7(U_1 + U_2 + U_3 - 5)$
8. with probability .0092, put $X = .7(U_1 + U_2 + U_3 + 2)$
9. with probability .0091347418, generate (x, y) uniformly from the rectangle of Fig. 2 until $y < g(x)$, then put $X = x$.
10. with probability .0004652582, generate pairs $x = 2U_1 - 1, y = U_2$ until $y < 3.5(12.25 - 21 \ln |x|)^{-.5}$ then put $X = \begin{cases} (12.25 - 2 \ln|x|)^{-.5} & \text{if } x < 0 \\ -(12.25 - 2 \ln|x|)^{-.5} & \text{if } x > 0 \end{cases}$

A CHI-SQUARE VARIATE

We will develop a procedure for generating a x^2_8 variate X , density $x^3 e^{-.5x}/96, x > 0$, along the same lines as the two examples above. We choose the interval $.4 \leq x \leq 20.4$ for fitting our mixture, dividing it into 10 parts. We will generate X in the form $2(M + U_1 + U_2 + U_3)$, where M takes values $.2, 1.2, 2.2, \dots, 7.2$ with probabilities p_1, p_2, \dots, p_8 . The best choice of the p 's comes from solving this problem:

Choose p_1, p_2, \dots, p_8 so as to maximize $p_1 + p_2 + \dots + p_8$

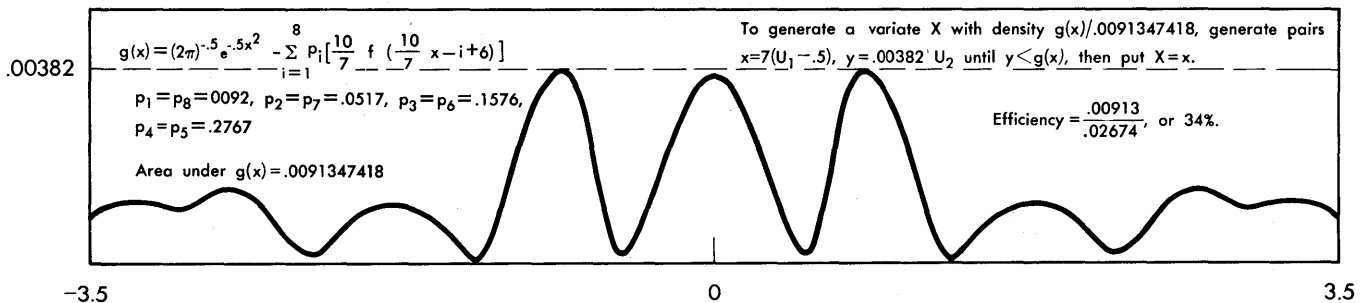


Figure 2. Method for generating a variate from the residual portion of the normal distribution.

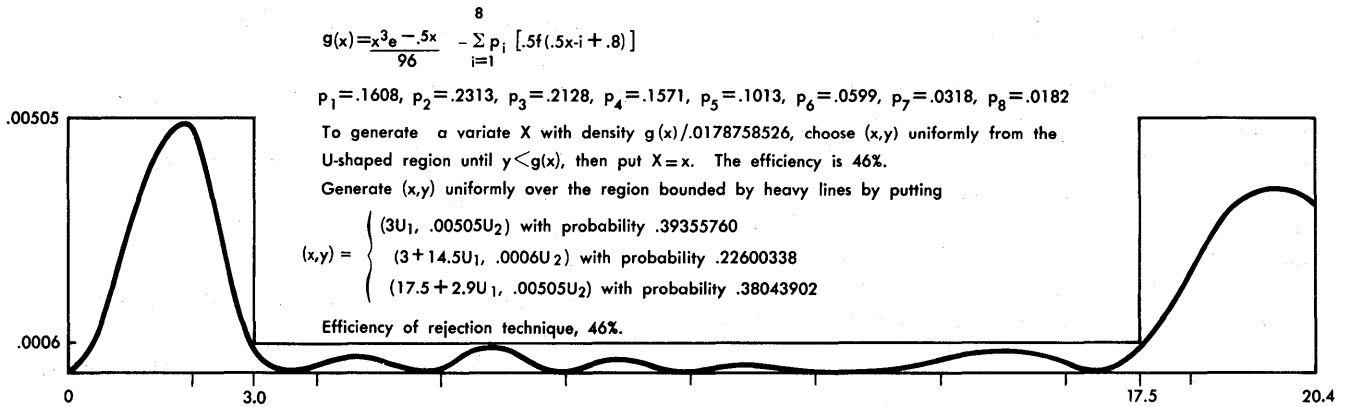


Figure 3. Method for generating a variate from the residual portion of the chi-square-8 distribution.

subject to the condition that $p_i > 0$ and

$$p_1 [.5f(.5x-2)] + p_2 [.5f(.5x-1.2)] + \dots + p_8 [.5f(.5x-7.2)] \leq x^3 e^{-.5x} / 96$$

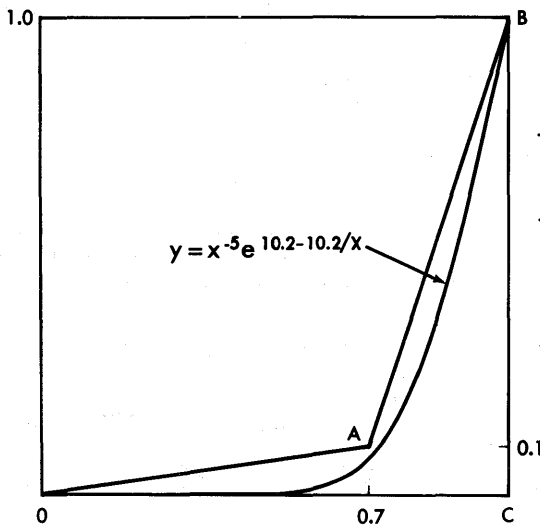
for $0 < x < 20.4$

The solution p_1, p_2, \dots, p_8 of this problem is given in the outline below and in Fig. 3. The sum of the p 's is .9732. We generate X from the residual density by the rejection technique as described in Fig. 3. We generate X from the tail, i.e., conditioned by $|X| > 20.4$, by transforming the tail to the unit interval and using the rejection technique (see Fig. 4). All of the steps combine to form this outline:

To generate a chi-square-8 variate X , density $x^3 e^{-.5x} / 96, x > 0$,

1. with probability $p_2 = .2313$ put $X = 2(1.2 + U_1 + U_2 + U_3)$

2. with probability $p_3 = .2128$ put $X = 2(2.2 + U_1 + U_2 + U_3)$
3. with probability $p_1 = .1608$ put $X = 2(.2 + U_1 + U_2 + U_3)$
4. with probability $p_4 = .1571$ put $X = 2(3.2 + U_1 + U_2 + U_3)$
5. with probability $p_5 = .1013$ put $X = 2(4.2 + U_1 + U_2 + U_3)$
6. with probability $p_6 = .0599$ put $X = 2(5.2 + U_1 + U_2 + U_3)$
7. with probability $p_7 = .0318$ put $X = 2(6.2 + U_1 + U_2 + U_3)$
8. with probability $p_8 = .0182$ put $X = 2(7.2 + U_1 + U_2 + U_3)$
9. with probability .0178758526 generate X from the residual density drawn in Fig. 3, by the rejection technique.



To generate a χ_8^2 variate X , conditioned by $X > 20.4$, choose (x,y) uniformly from the quadrilateral OABC until $y < x^{-5} e^{10.2-10.2/x}$, then put $X=20.4/x$. The efficiency is 70%. Generate (x,y) uniformly from OABC by putting

$$(x,y) = \begin{cases} (.7-.7M+m, .1-1M) & \text{with probability } 1/4 \\ (.7+.3M, .1+.9M-m) & \text{with probability } 3/4, \end{cases}$$

where $M = \max(U_1, U_2)$ and $m = \min(U_1, U_2)$.

Figure 4. Method for generating a variate from the tail of the chi-square-8 distribution.

10. with probability .0089241474 generate X from the tail of the x_s^2 distribution, by the rejection technique described in Fig. 4.

GENERAL REMARKS

The examples above suggest the following general procedure for dealing with a density $q(x)$. The only requirement is that q be close to the x -axis at its extremities. An interval containing most of the density is chosen, say $a < x < b$, then divided into n equal parts; 10 is usually a good choice. If $h = (b-a)/10$, then this linear programming-type problem is solved: choose p_1, \dots, p_8 so as to maximize $p_1 + p_2 + \dots + p_8$, subject to the condition that $p_i > 0$ and

$$p_1 f\left(\frac{x-a}{h}\right) + p_2 f\left(\frac{x-a-h}{h}\right) + \dots + p_8 f\left(\frac{x-a-7h}{h}\right) \leq hq(x)$$

Then X may be generated by putting $X = a + h(M + U_1 + U_2 + U_3)$, where M takes values $0, 1, 2, \dots, 7$ with probabilities p_1, \dots, p_8 , or by choosing X from the residual density by the rejection technique, or from the tail, in a manner suggested by the above examples. The sum of the p 's will usually be quite close to 1—it was .9915, .9904, and .9732 in the three examples, and thus the resulting programs will be very fast. Few constants are needed, and the programs should be easy to code; they vary little from one density to the next, only the constants and some

details from the residues or tails changing. In fact, the fast parts of the programs—generating $c(M + U_1 + U_2 + U_3)$, are so consistent from one density to the next that a basic program for this part of the outline can be written in machine language, with the constants inserted for the particular density under consideration. The slow parts of the program—the residual density and tail can be handled by FORTRAN, or some such convenient language, subroutines.

The procedure for a normal variate outlined above is almost as fast as the super program in Ref. 3, yet it is much simpler and requires very little computer space.

REFERENCES

1. M. D. MacLaren, G. Marsaglia and T. A. Bray, "A Fast Procedure for Generating Exponential Variables," *Communications of the Association for Computing Machinery*, vol. 7, no. 5 (1964).
2. G. Marsaglia, "Expressing a Random Variable in Terms of Uniform Random Variables," *Annals of Mathematical Statistics*, vol. 32, pp. 894–98 (1961).
3. —, M. D. MacLaren and T. A. Bray, "A Fast Procedure for Generating Normal Variables," *Communications of the Association for Computing Machinery*, vol. 7, no. 1 (1964).
4. —, "Generating a Variable from the Tail of the Normal Distribution," *Technometrics*, vol. 6, no. 1, pp. 101–2 (1964).

A UNIFIED APPROACH TO DETERMINISTIC AND RANDOM ERRORS IN HYBRID LOOPS

Jacques J. Vidal

*University of California
Los Angeles, California*

INTRODUCTION

Hybrid computation in general owes its growing acceptance to the increasing number of sophisticated problems that neither digital nor analog computers alone can handle adequately. In most cases this apparently cumbersome approach is justified by the speed of analog computers, a speed impossible to attain in all-digital systems, even where the problem requires the memory and logic capabilities that only digital computers can provide.

Both types of computers introduce specific errors in the hybrid loop. In addition, a number of perturbations are generated by the exchange and distribution of the information between the computers and by analog-digital and digital-analog conversion. These "interface errors" must be considered when evaluating a particular hybrid configuration, as they may become a major source of solution errors. A breakdown of the error sources in a typical loop is represented in Fig. 1. Many of them result from a tradeoff between performance and equipment complexity. For instance, parallel transfer of information can remove the necessity for multiplexing and distribution. In addition, quantization and round-off errors may be made negligible by increasing the number of bits used to represent digital information.

It is clear from Fig. 1 that most of the error sources are random rather than deterministic. On the

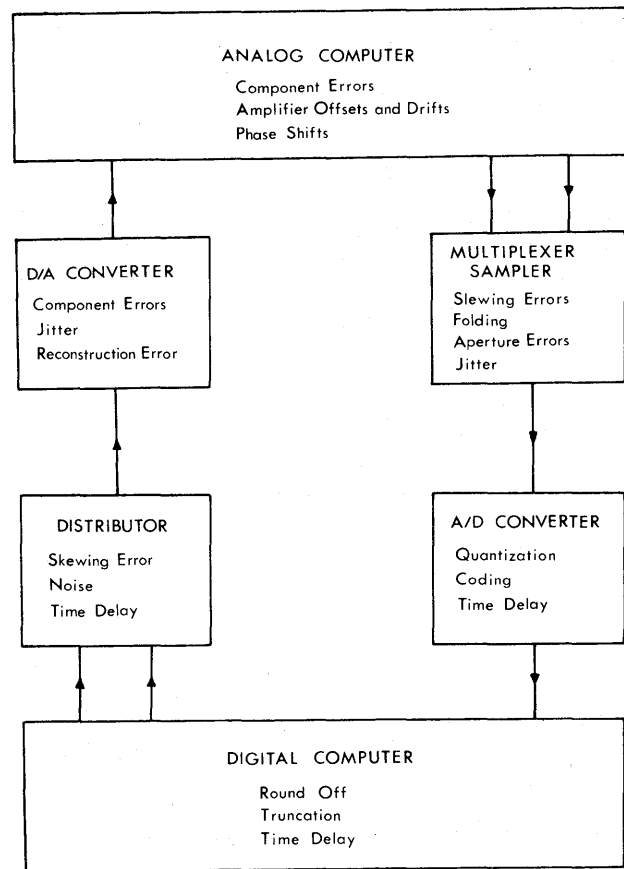


Figure 1. Typical hybrid loop with major error sources.

other hand, deterministic errors frequently lend themselves to some compensation. A previous paper was devoted to deterministic errors and presented a method, based on sensitivity analysis, that provided the solution error functions under some general conditions.

The aim of the present study is to extend the same approach to random perturbations, as suggested by Meissinger² in the particular case of noise excitations. In the process, uncompensated deterministic errors will be included as a special case. A statistical (standard deviation) bound will be provided for the solution error, regardless of the deterministic nature of some of the incident perturbations. It is first recognized that a dynamic system under study on the hybrid computer may be represented by a system of first-order linear or nonlinear differential equations:

$$\dot{x} = f(x,t) \quad x(0) = x_0 \quad (1)$$

where $x = \{x_n\}$; $n = 1, 2, \dots, N$ is the state N -vector; and x_0 the initial state. The trajectory in the state space

$$x = x(t) \quad 0 \leq t \leq T$$

is the nominal (error-free) solution. The first step in the present error analysis scheme is the identification of the perturbations and the introduction of a corresponding additional forcing function into system (1). This task is generally straightforward and involves transforming (1) into a perturbed system of equations. This is sometimes done by Taylor expansion, in which case only first-order terms are retained. The perturbed equations take the form:

$$\begin{aligned} \dot{x} &= f(x,t) + H(x,t) \cdot q \\ 0 &\leq t \leq T \\ x(0) &= x_0 \end{aligned} \quad (2)$$

where $q = \{g_j\}$; and $j = 1, 2, \dots, J$ is a J -vector whose elements are weighting factors characterizing the intensity of the J error sources acting on the system.

$$H = \{h_{nj}(x,t)\}$$

is a $N \times J$ matrix whose elements are the functions of the state variables and time, coupling each error source j to each state equation. The individual functions h_{nj} can frequently be found by inspection or by a first-order expansion of the function f .

The presence of the additional forcing terms causes the solution of (2) to differ from the nominal solution $x(t)$. This may be written:

$$x(t) + \Delta x(t) \quad (3)$$

where $\Delta x = \{\Delta x_n\}$ is the state deviation N -vector. Two major assumptions are made at this point:

1. It is assumed that the perturbed solution lies in a linear domain in the state space in the vicinity of the nominal solution. Then the deviations due to individual perturbations are additive and

$$\Delta x_n(t) = \sum_{j=1}^J \Delta x_{nj}(t) \quad (4)$$

where Δx_{nj} is the contribution of the perturbation j to Δx_n .

2. It is assumed that the state deviations Δx_{nj} depend linearly on the intensity of the perturbation j . Then

$$\Delta x_{nj}(t) = u_{nj}(x,t) \cdot q_j \quad (5)$$

and there exists a set of $N \times J$ functions u_{nj} (sensitivity functions) that linearly relate the state deviations to the intensity of the error sources. They form a $N \times J$ matrix

$$U = \{u_{nj}(x,t)\}$$

Both of these assumptions depend on the fact that errors must remain small through the integration domain if the computed solution is to have any value. They imply that $f(x,t)$ must be sufficiently smooth and q sufficiently small.

Analyzing the error sources that generate the additional terms introduced in (2) reveals that the perturbations can be separated into four classes in accordance with the deterministic or stochastic character of the forcing function and the deterministic or random nature of the weighting coefficient. The first class covers systematic and known error sources in the computing system. The average time delay in the digital loop is of that type.³ The second class corresponds to error sources that are systematic in nature but whose amplitude is only statistically known. Departures from nominal value in the analog components are one example: The distribution of the actual values in the component supply may be known or hypothesized, but actual deviations in the particular elements used are unknown. Another example is provided by the offsets or drifts in analog amplifiers, which have a constant value during a computer run but are known only by the equipment specifications and performances. For the third and fourth classes the forcing functions $h_{nj}(x,t)$ are stochastic processes. The third class covers perturbations that have a noise-like character and fluctuate continuously during the solution but are

systematic in nature such as round-off, quantization and reconstruction errors. Although there is an alternate and more rigorous deterministic approach to those, the statistical description is in general sufficient to provide adequate error estimates. Finally, perturbations arise where both the forcing function and the weighting coefficient are random. Incident noise and jitter are important examples.

At this point, the following hypotheses are made:

1. When q_j is random it has zero mean. This is always realizable with a proper choice of q_j .

2. When $h_{nj}(x,t)$ is stochastic, it takes the approximate form:

$$h_{nj}(x,t) = \tilde{h}_{nj}(x,t)n(t) \quad (6)$$

where \tilde{h}_{nj} is deterministic and $n(t)$ is white noise of unity power density. Then h_{nj} has zero mean and the actual power density of the perturbation must be characterized by q_j^2 .

3. h_{nj} and q_j are statistically independent (or at least uncorrelated) for all t .

In the section below on the derivation of the sensitivity equations, the sensitivity functions u_{nj} are shown to constitute the outputs of linear differential equations whose input forcing functions are h_{nj} . Therefore when h_{nj} is given by (6), u_{nj} has zero mean and for any t , q_j and u_{nj} are independent random variables. With these assumptions, taking the expected values of both sides in relation (5) yields

$$\begin{aligned} E\{\Delta x_{nj}\} &= E\{u_{nj}q_j\} \\ &= E\{u_{nj}\}E\{q_j\} \end{aligned} \quad (7)$$

$$\begin{aligned} E\{\Delta x_{nj}^2\} &= E\{u_{nj}^2q_j^2\} \\ &= E\{u_{nj}^2\}E\{q_j^2\} \end{aligned} \quad (8)$$

Finally the values or statistical moments of q_j , u_{nj} and Δx_{nj} may be summarized (Table 1).

The weighting factors q_j are either known constants or random variables of known distribution $w(q_j)$, in which case $q_j^2 = \int_{-\infty}^{\infty} q_j^2 w(q_j) dq_j$. The sensitivity functions u_{nj} (or u_{nj}^{-2}) are computed according to a procedure discussed in the section on derivations. Therefore, as shown in Table 1, the quantities $E\{\Delta x_{nj}\}$ and $E\{\Delta x_{nj}^2\}$ are completely determined.

ESTIMATION OF THE SOLUTION ERROR

Let ϵ^2 denote the mean square of the length $\|\Delta x\|$ of the state deviation vector in presence of J perturbation sources:

$$\epsilon^2(t) = E\{\|\Delta x(t)\|^2\} \quad (9)$$

A bound for ϵ^2 , expressed in terms of the known elements in the last column of Table 1, is provided by the following

Theorem:

If $\Delta x = \{\Delta x_n\}$ is an N -vector whose components are given by relation (4), then, for all t , the following inequality holds:

$$\begin{aligned} \epsilon^2 &\leq \sum_{n=1}^N \left(\sum_{j=1}^J E\{\Delta x_{nj}^2\} \right. \\ &\quad \left. + 2 \sum_{j=1}^{J-1} \sum_{k=j+1}^J \sqrt{E\{\Delta x_{nj}^2\}E\{\Delta x_{nk}^2\}} \right) \end{aligned} \quad (10)$$

Proof: The squared length of Δx is given by

$$\begin{aligned} \|\Delta x\|^2 &= \sum_{n=1}^N \Delta x_n^2. \text{ Taking the expected values} \\ \epsilon^2 &= E \left\{ \sum_{n=1}^N \Delta x_n^2 \right\} = \sum_{n=1}^N E \left\{ \Delta x_n^2 \right\} \end{aligned} \quad (11)$$

$E\{\Delta x_n^2\}$ may be evaluated by replacing Δx_n by its value given by (4):

Table 1

Class	Weighting Factor			Sensitivity Function			Deviation	
	q_j	$E\{q_j\}$	$E\{q_j^2\}$	u_{nj}	$E\{u_{nj}\}$	$E\{u_{nj}^2\}$	$E\{\Delta x_{nj}\}$	$E\{\Delta x_{nj}^2\}$
1	const.	q_j	q_j^2	determ.	u_{nj}	u_{nj}^2	$q_j u_{nj}$	$q_j^2 u_{nj}^2$
2	rand	0	q_j^2	determ.	u_{nj}	u_{nj}^2	0	$q_j^2 u_{nj}^2$
3	const.	q_j	q_j^2	stochas.	0	u_{nj}^{-2}	0	$q_j^2 u_{nj}^{-2}$
4	rand.	0	q_j^2	stochas.	0	u_{nj}^{-2}	0	$q_j^2 u_{nj}^{-2}$

$$E\{\Delta x_n^2\} = E\left\{\left(\sum_{j=1}^J \Delta x_{nj}\right)^2\right\}$$

$$= E\left\{\sum_{j=1}^J \Delta x_{nj}^2 + 2 \sum_{j=1}^{J-1} \sum_{k=j+1}^J \Delta x_{nj} \Delta x_{nk}\right\}$$

or

$$(12)$$

$$E\{\Delta x_n^2\} = \sum_{j=1}^J E\{\Delta x_{nj}^2\} + 2 \sum_{j=1}^{J-1} \sum_{k=j+1}^J E\{\Delta x_{nj} \Delta x_{nk}\}$$

Using the Schwarz inequality

$$E\{\Delta x_{nj} \Delta x_{nk}\} \leq \sqrt{E\{\Delta x_{nj}^2\} E\{\Delta x_{nk}^2\}} \quad (13)$$

Then

$$E\{\Delta x_n^2\} \leq \sum_{j=1}^J E\{\Delta x_{nj}^2\} + 2 \sum_{j=1}^{J-1} \sum_{k=j+1}^J \sqrt{E\{\Delta x_{nj}^2\} E\{\Delta x_{nk}^2\}} \quad (14)$$

Replacing in (11) establishes the theorem.

Quite obviously, the same relation (10) provides also a bound for the mean

$$\mu = E\{\Delta x\} \leq \varepsilon \quad (15)$$

since the variance ($\varepsilon^2 - \mu^2$) is always positive.

DERIVATION OF THE SENSITIVITY EQUATIONS

A differential equation in U may be obtained by differentiating (2) with respect to q and permuting the differentiation symbols. The latter operation is permissible since the solution is assumed continuously dependent on q :

$$\dot{U} = \left[\frac{\partial f(x,t)}{\partial x} + q \cdot \frac{\partial f(x,t)}{\partial x} \right]_{q=0} U + H(x,t)$$

then

$$\dot{U} = G(x,t) \cdot U + H(x,t) \quad U(0) = 0 \quad (16)$$

where $G(x,t) = \frac{\partial f(x,t)}{\partial x}$ is a $N \times N$ Jacobian matrix.

Since no error may have developed at $t = 0$, it may be seen from (5) that all initial conditions for (16) are zero. In compact notation, the matrix Eq. (16) represents J -independent systems of N linear, time-dependent differential equations (one for each column; that is, one for each perturbation j). Taking

one system at a time, (16) reduces to the vector equation:

$$\dot{u}_j = G(x,t) u_j + h_j \quad (17)$$

where $u_j = \{u_{nj}\}$ and $h_j = \{h_{nj}\}$, $n = 1, 2, \dots, N$, are N -vectors. The remarkable point about (17) is that its homogeneous part is the same for all j . Therefore, if the transition matrix of (17) is known, superposition techniques may be applied for each forcing function. This is a major advantage when a large number of perturbations are present since a unique transition matrix has to be computed. The computational aspects of generating the transition matrix and the subsequent mean-square sensitivities are discussed below and in the next paragraph. A classical use is made of the adjoint homogeneous system that provides the transition matrix in convenient form. It must be noted that the digital storage capabilities of hybrid computers remove the simulation difficulties encountered when this approach is implemented with pure analog techniques. The superposition integral that leads to the mean-square sensitivities may be evaluated digitally afterwards. In the latter operation, aimed directly at the generation of the error bounds, accuracy is usually unimportant. A crude integration technique and large sampling intervals are generally acceptable. The homogeneous equation associated with (17) is

$$\dot{v}(t) = G(x,t)v(t) \quad (18)$$

Let $\phi(t,\tau)$ be the state transition matrix for (18).

Then

$$u_j(t) = \int_0^t \phi(t,\tau) \cdot h_j(x,\tau) \cdot d\tau$$

and

$$u_j(T) = \int_0^T \phi(T,\tau) \cdot h_j(x,\tau) \cdot d\tau \quad (19)$$

The state transition matrix may be obtained experimentally by solving (18), since $\phi(t,\tau)$ satisfies by definition

$$\dot{\phi}(t,\tau) = G(x,t) \phi(t,\tau) \quad \phi(t,\tau) = I \quad (20)$$

Computing the matrix $\phi(t,\tau)$ implies N integration of the system, taking each column of the identity matrix in turn for initial conditions at $t = \tau$, and therefore obtaining, per run, one column of the transition matrix for that particular τ . Since (19) requires that $\phi(t,\tau)$ be available as a function of τ for

a fixed $t = T$, it is advantageous to solve the adjoint system instead. The adjoint to (18) becomes

$$\dot{w}(t) = -G^*(x, t) w(t) \quad (21)$$

where G^* designates the transpose of the matrix G . The state transition matrix for (21) is $\psi(t, \tau)$, and satisfies

$$\begin{aligned} \dot{\psi}(t, \tau) &= -G^*(x, t) \cdot \psi(t, \tau) \\ \psi(\tau, \tau) &= I \end{aligned} \quad (22)$$

On the other hand, the following relation exists between ψ and ϕ^* :

$$\begin{aligned} \psi(t, \tau) &= \phi^*(\tau, t) \\ \text{or letting } \tau &= T \\ \psi(t, T) &= \phi^*(T, t) \end{aligned} \quad (23)$$

Therefore, solving one column of system (22) with $\tau = T$ will yield one row of the functions ϕ as they appear in (11). However, this would be a final value problem with the condition

$$\psi(T, T) = I \quad t \leq T$$

A classical change of variable yields the so-called modified adjoint system, where the final conditions are replaced by initial conditions more suitable for computing:

$$t = T - \zeta$$

The new variable ζ runs backward with respect to the problem time in the interval $(0, T)$. Putting $\tau = T$ in (22) and transforming yields:

$$\dot{\psi}(T - \zeta, 0) = G^*(x, T - \zeta) \cdot \psi(T - \zeta, 0) \quad (24)$$

which can be solved as an initial value problem with initial conditions obtained columnwise from the identity matrix:

$$\psi(0, 0) = I$$

It must be noted that the function $x(t)$ appearing in G^* must also be introduced backward as $x(T - \zeta)$. This presents no computing difficulty on the hybrid computer as long as one complete run of the original system in the interval $(0, T)$ is available from memory.

EVALUATION OF THE MEAN-SQUARE SENSITIVITIES

At this point, for any deterministic vector h_j (classes 1 and 2), actual values of the sensitivity function at $t=T$ may be computed according to Eq. (19). With the change in independent variable, this integral becomes:

$$u_j(T) = \int_0^T \psi^*(T - \zeta, 0) \cdot h_j(T - \zeta) d\zeta \quad (25)$$

The procedure is not applicable when h_j is a stochastic vector function. However, when the elements h_{nj} have the form given by (7), the autocorrelation matrix $A(u_j)$ can be evaluated by

$$A(u_j) = u_j > < u_j^* = \int_0^T \psi^*(T - \zeta, 0) H_j(T - \zeta) \psi(T - \zeta, 0) d\zeta \quad (26)$$

where $A(u_j) = \{u_{nj} \cdot u_{mj}\} n = 1, 2 \dots N; m = 1, 2, \dots, N$, and H_j is the outer product

$$H_j = \tilde{h}_{nj} > < \tilde{h}_{nj}^*, n = 1, 2, \dots, N$$

whose elements are the deterministic part of the perturbation functions. The diagonal terms of $A(u_j)$ are the mean-square values of the sensitivity function. Expanding (26) yields

$$\overline{u_{nj}^2(T)} = \int_0^T \sum_{m=1}^N h_{mj}^2(T - \zeta) \psi_{nm}^2(T - \zeta, 0) d\zeta \quad (27)$$

The flow chart given in Fig. 2 illustrates the procedure.

CONCLUSION

The method may now be summarized. A first computer run is used to solve the original system (1) and store the N output functions in tabular form. The stored x -functions are then read out backward to drive the modified adjoint system (24). This operation requires N separate runs as the initial conditions are rotated and yields the N^2 functions

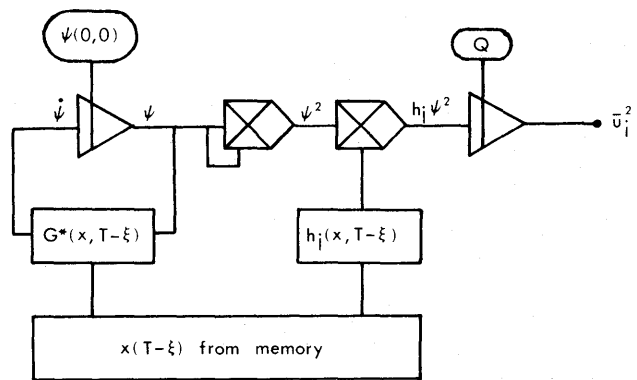


Figure 2. Symbolic flow chart for the mean-square sensitivities.

$(T-\zeta, 0)$ that constitute the elements of the transition matrix. This information is now used to evaluate the superposition integrals (25) or (27). Only at this stage are the specific forcing functions $h_{nj}(x, t)$ introduced. A moderate accuracy is generally acceptable for the integration and memory space may be saved by a coarse sampling of the functions ψ_{nm} and h_{nj} . The functions ψ_{nm} are independent of any specific perturbation and the procedure becomes especially economical when a large number of error sources are to be considered. Also, critical error sources may be singled out to direct eventual corrective action.

Finally, the bound on the total mean square deviation may be computed in accordance with (10) after forming the proper products of the mean-square sensitivities and weighting factors.

REFERENCES

1. W. J. Karplus and J. Vidal, "Characterization and Evaluation of Hybrid Systems," *Proceedings IFAC Symposium on System Engineering*, Tokyo, Aug. 1965.
2. H. Meissinger, "Parameter Influence Coefficients and Weighting Functions Applied to Perturbation Analysis of Dynamic Systems," *Proceedings Third International Congress AICA*, Optija, Yugoslavia, 1961, pp. 207-16.
3. R. Tomovic, W. J. Karplus and J. Vidal, "Sensitivity of Discrete-Continuous Systems," *Proceedings Third Congress of the IFAC*, London, June 20-25, 1966.
4. T. Miura and J. Iwata, "Effects of Digital Execution Time in a Hybrid Computer," *Proceedings FJCC*, 1963, pp. 251-66.
5. J. Vidal, W. J. Karplus and G. Keludjian, "Sensitivity Coefficients for the Correction of Quantization Errors in Hybrid Computer Systems," *Proceedings IFAC Symposium on Sensitivity Analysis*, Dubrovnik, Yugoslavia, Sept. 1964 (L. Radanovic, ed.), Pergamon Press, Oxford, 1966, pp. 197-208.
6. —, and W. J. Karplus, "Characterization and Compensation of Quantization Errors in Hybrid Computer Systems," *IEEE International Convention Record*, New York, 1965, pt. III, pp. 236-41.
7. J. H. Laning, Jr., and R. H. Battin, *Random Processes in Automatic Control*, McGraw-Hill, New York, 1956.
8. G. A. Korn, *Random-Process Simulation and Measurements*, McGraw-Hill, New York, 1965.

HYBRID COMPUTER SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS BY MONTE CARLO METHODS

Warren D. Little

*Process Computer Engineering, Canadian General Electric Company
Peterborough, Ontario*

INTRODUCTION

In addition to finite-difference methods,¹ Monte Carlo methods² also are known for solving certain partial differential equations. When implemented on a digital computer, however, the Monte Carlo methods have generally proven to be very inefficient. In 1960, a study carried out at the University of Michigan described analog computer techniques for mechanizing Monte Carlo methods.³ From the Michigan study it became evident that a fast analog computer together with a small digital computer and a modest interface could obtain Monte Carlo solutions at rates competitive with standard finite-difference methods.

With rare exception, the Monte Carlo methods that have been programmed on either an analog or digital computer have been for elliptic partial differential equations (e.g., Laplace's equation). In this paper the classical Monte Carlo methods are generalized to yield methods for solving parabolic equations (e.g., the diffusion equation) as well as homogeneous and nonhomogeneous elliptic equations of a very general form.⁴ Techniques for implementing the Monte Carlo methods on a hybrid system consisting of a general-purpose analog computer and a general-purpose digital computer as well as some typical results are also discussed.

PARTIAL DIFFERENTIAL EQUATIONS FOR CONTINUOUS MARKOV PROCESSES

The Monte Carlo methods to be discussed are based upon partial differential equations that can be written for continuous Markov processes. A continuous Markov process is a stochastic process having the property that future values of a stochastic variable \bar{r} depend only upon present and not past values. To solve a boundary value problem that is defined on an open bounded region R and its boundary C , a continuous Markov process defined on R and C must be simulated.

To facilitate description, consider a three-dimensional Markov process with stochastic vector $\bar{r} = \bar{r}(x,y,z)$ where components x , y and z are given by the following stochastic differential equations:

$$\frac{dx}{dt} + A_1(x,y,z,t) = B_1(x,y,z,t)N_1(t) \quad (1)$$

$$\frac{dy}{dt} + A_2(x,y,z,t) = B_2(x,y,z,t)N_2(t) \quad (2)$$

$$\frac{dz}{dt} + A_3(x,y,z,t) = B_3(x,y,z,t)N_3(t) \quad (3)$$

The coefficients A_i and B_i are, in general, slowly varying continuous functions of x,y,z and t . The driving terms $N_i(t)$ are uncorrelated with each other

and each term is ideally Gaussian white noise with power spectral density $2D_i$.

For a Markov process as defined by Eqs. (1) to (3), two relevant conditional probability density functions f and g are defined.

(A) $f(\bar{r}_2, t_2 | \bar{r}_0, t_0) dr_2$ is the probability that the stochastic vector \bar{r} is in dr_2 within C at time t_2 if at time t_0 , $\bar{r} = \bar{r}_0$.

(B) $g(\bar{r}_b, t_b | \bar{r}_0, t_0) dr_b dt_b$ is the probability that the stochastic vector \bar{r} will reach boundary C for the first time within dr_b between times t_b and $t_b + dt_b$ if at time t_0 , $\bar{r} = \bar{r}_0$.

The notation used in definitions (A) and (B) is illustrated in Fig. 1.

The conditional probability density functions defined above satisfy the following initial and boundary conditions

$$(A) \quad \lim_{t_0 \rightarrow t_2} f(\bar{r}_2, t_2 | \bar{r}_0, t_0) = \delta(\bar{r}_2 - \bar{r}_0)$$

where $\delta(\bar{r}_2 - \bar{r}_0) = 0$ for $\bar{r}_2 \neq \bar{r}_0$

$$\text{and } \int_R \delta(\bar{r}_2 - \bar{r}_0) dr_2 = 1.$$

$$(B) \quad \lim_{t_0 \rightarrow t_2} g(\bar{r}_b, t_b | \bar{r}_0, t_0) = 0$$

$$(C) \quad \lim_{\bar{r}_0 \rightarrow \bar{r}_b} f(\bar{r}_2, t_2 | \bar{r}_0, t_0) = 0$$

$$(D) \quad \lim_{\bar{r}_0 \rightarrow \bar{r}_b} g(\bar{r}_b, t_b | \bar{r}_0, t_0) = \delta(\bar{r}_0 - \bar{r}_b) \delta(t_b - t_0)$$

where $\delta(\bar{r}_0 - \bar{r}_b) \delta(t_b - t_0) = 0$ unless $\bar{r}_0 = \bar{r}_b$ and

$$t_b = t_0, \text{ and } \int_{t_0}^{t_2} \int_C \delta(\bar{r}_0 - \bar{r}_b) \delta(t_b - t_0) dr_b dt_b = 1$$

For the above Markov process it can be shown that the density functions f and g also satisfy the following backward Kolmogorov partial differential equations:^{4,5}

$$-\frac{\partial f(\bar{r}_2, t_2 | \bar{r}_0, t_0)}{\partial t_0} = L_{\bar{r}_0, t_0} f(\bar{r}_2, t_2 | \bar{r}_0, t_0) \quad (4)$$

$$-\frac{\partial g(\bar{r}_b, t_b | \bar{r}_0, t_0)}{\partial t_0} = L_{\bar{r}_0, t_0} g(\bar{r}_b, t_b | \bar{r}_0, t_0) \quad (5)$$

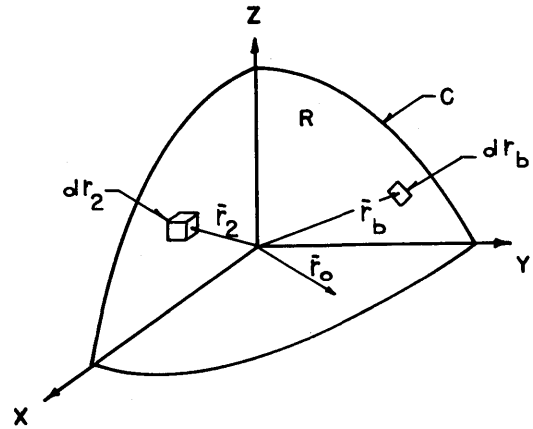


Figure 1. Space region illustrating terms used in definitions.

where

$$L_{\bar{r}_0, t_0} = a_1(\bar{r}_0, t_0) \frac{\partial}{\partial x_0} + a_2(\bar{r}_0, t_0) \frac{\partial}{\partial y_0} + a_3(\bar{r}_0, t_0) \frac{\partial}{\partial z_0} + b_1(\bar{r}_0, t_0) \frac{\partial^2}{\partial x_0^2} + b_2(\bar{r}_0, t_0) \frac{\partial^2}{\partial y_0^2} + b_3(\bar{r}_0, t_0) \frac{\partial^2}{\partial z_0^2} \quad (6)$$

The coefficients $a_i(\bar{r}_0, t_0)$ and $b_i(\bar{r}_0, t_0)$ are moments of the Markov process and are related to the coefficients A_i and B_i of the stochastic differential equations by Eqs. (7) and (8).

$$a_i(\bar{r}_0, t_0) = -A_i(\bar{r}_0, t_0) \quad (7)$$

$$b_i(\bar{r}_0, t_0) = [B_i(\bar{r}_0, t_0)]^2 D_i \quad (8)$$

By simulating the Markov process given by Eqs. (1) to (3), boundary value problems with partial differential equations of the same form as Eqs. (4) and (5) can be solved. More general partial differential equations in which the dependent variable and not only its derivative exit can also be solved by considering auxiliary probability density functions u and v as defined below.^{4,6}

$$u(\bar{r}_2, t_2 | \bar{r}_0, t_0) = \exp -m(t_2, t_0) \begin{cases} \bar{r}(t_2) = \bar{r}_2 \\ \bar{r}(t_0) = \bar{r}_0 \end{cases} f(\bar{r}_2, t_2 | \bar{r}_0, t_0) \quad (9)$$

$$v(\bar{r}_b, t_b | \bar{r}_o, t_o) = \exp -m(t_b, t_o) \begin{cases} \bar{r}(t_b) = \bar{r}_b \\ \bar{r}(t_o) = \bar{r}_o \end{cases} g(\bar{r}_b, t_b | \bar{r}_o, t_o) \quad (10)$$

In these definitions the brackets $\left. \begin{matrix} \bar{r}(t_2) = \bar{r}_2 \\ \bar{r}(t_o) = \bar{r}_o \end{matrix} \right\}$

denote a conditional expectation; that is, the expected value of the function within the brackets subject to the condition that $\bar{r}(t_o) = \bar{r}_o$ and $\bar{r}(t_2)$ is in a small region dr_2 about \bar{r}_2 . The term $m(t_2, t_o)$ is the integral of a slowly varying continuous positive function $d(\bar{r}(t), t)$ of the stochastic vector $\bar{r}(t)$ and t . That is,

$$m(t_2, t_o) = \int_{t_o}^{t_2} d(\bar{r}(t), t) dt \quad (11)$$

The auxiliary probability density functions u and v satisfy the following partial differential equations.

$$\begin{aligned} \frac{\partial v}{\partial t_o} (\bar{r}_b, t_b | \bar{r}_o, t_o) &= L_{\bar{r}_o, t_o} u(\bar{r}_2, t_2 | \bar{r}_o, t_o) \\ &\quad - d(\bar{r}_o, t_o) u(\bar{r}_2, t_2 | \bar{r}_o, t_o) \end{aligned} \quad (12)$$

$$\begin{aligned} \frac{\partial v}{\partial t_o} (\bar{r}_b, t_b | \bar{r}_o, t_o) &= L_{\bar{r}_o, t_o} v(\bar{r}_b, t_b | \bar{r}_o, t_o) \\ &\quad - d(\bar{r}_o, t_o) v(\bar{r}_b, t_b | \bar{r}_o, t_o) \end{aligned} \quad (13)$$

Equations (4), (5), (12) and (13), together with initial and boundary conditions (A) to (D), will now be used to develop Monte Carlo methods for solving a large class of partial differential equations.

MONTE CARLO METHODS FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS

In this section, relationships between probability density functions f , g , u and v and solutions ϕ of boundary value problems will be developed. The methods apply to problems in which ϕ itself is given initially and on all boundaries. For all problems the solution at a point within a region R is obtained as the expected value of initial and boundary values at the terminal points of random walks that originate at the point for which the solution is desired. The expected value is written in terms of the functions f ,

g , u and v . An approximation to the expected value is determined experimentally from a large number of random walks simulated on an analog computer.

Consider the following boundary value problem involving a parabolic partial differential equation.

Problem A

Determine $\phi(\bar{r}_o, t_o)$ such that:

$$(1) \quad \frac{\partial \phi(\bar{r}_o, t_o)}{\partial t_o} = L_{\bar{r}_o, t_o} \phi(\bar{r}_o, t_o) \quad (14)$$

is satisfied within a bounded region R ;

(2) a piecewise continuous initial condition $\phi_o(\bar{r}_o)$ is satisfied within R ; i.e.,

$$\phi(\bar{r}_o, 0) = \phi_o(\bar{r}_o) \quad (15)$$

(3) a piecewise continuous boundary condition $\phi_c(\bar{r}_b, t_b)$ is satisfied on the boundary C of R ; i.e.,

$$\phi(\bar{r}_b, t_b) = \phi_c(\bar{r}_b, t_b). \quad (16)$$

The boundaries and initial and boundary conditions of a typical problem with one space variable are shown in Fig. 2. Note that $\phi(\bar{r}_o, t_o)$ is a function of the initial position \bar{r}_o and starting time t_o . The time t_o is defined to be negative so that random walks take place in the time interval $t_o \leq t \leq 0$.

To obtain the solution ϕ of Problem A at a point (\bar{r}_o, t_o) , random walks are started at (\bar{r}_o, t_o) and each walk is terminated as soon as a boundary is reached or at $t = 0$. If a walk terminates on a boundary at some (\bar{r}_b, t_b) the boundary value $\phi_c(\bar{r}_b, t_b)$ is recorded, whereas if a walk is terminated at $t = 0$ with position \bar{r}_2 , the initial value $\phi_o(\bar{r}_2)$ is recorded. The expected value of the initial and boundary values obtained in this manner is a solution of Problem A.

The expected value defined above can be written in terms of density functions f and g as follows:

$$\begin{aligned} \phi(\bar{r}_o, t_o) &= \int_R \phi_o(\bar{r}_2) f(\bar{r}_2, 0 | \bar{r}_o, t_o) dr_2 \\ &\quad + \int_{t_o}^0 \int_C \phi_c(\bar{r}_b, t_b) g(\bar{r}_b, t_b | \bar{r}_o, t_o) dr_b dt_b \end{aligned} \quad (17)$$

The fact that $\phi(\bar{r}_o, t_o)$ as given above is indeed a solution of the problem can be shown by operating on both the right and left side of Eq. (17) with the

operator $\frac{\partial}{\partial t_o} + L_{\bar{r}_o, t_o}$. This gives

$$\begin{aligned}
 \left(\frac{\partial}{\partial t_0} + L_{\bar{r}_0, t_0} \right) \phi(\bar{r}_0, t_0) = & \\
 & \int_R \phi_0(\bar{r}_2) \left(\frac{\partial}{\partial t_0} + L_{\bar{r}_0, t_0} \right) f(\bar{r}_2, 0 | \bar{r}_0, t_0) dr_2 \\
 & + \int_{t_0}^0 \int_C \phi_c(\bar{r}_b, t_b) \left(\frac{\partial}{\partial t_0} + L_{\bar{r}_0, t_0} \right) g(\bar{r}_b, t_b | \bar{r}_0, t_0) dr_b dt_b \\
 & - \int_C \phi_c(\bar{r}_b, t_0) g(\bar{r}_b, t_0 | \bar{r}_0, t_0) dr_b
 \end{aligned} \tag{18}$$

The right side of Eq. (18) is zero by Kolmogorov Eqs. (4) and (5) and initial condition (B). Thus $\phi(\bar{r}_0, t_0)$ satisfies the partial differential equation of Problem A.

By direct application of initial and boundary conditions (A) to (D) to Eq. (17), it also follows that $\phi(\bar{r}_0, t_0)$ satisfies the initial and boundary conditions of Problem A.

The Monte Carlo solution of Problem A is obtained by approximating the expected value $\phi(\bar{r}_0, t_0)$ given by Eq. (17) with the average $\phi_N(\bar{r}_0, t_0)$ of initial and boundary values ϕ_i that are recorded from a set of N random walks originating at (\bar{r}_0, t_0) . This average is

$$\phi_N(\bar{r}_0, t_0) = \frac{1}{N} \sum_{i=1}^N \phi_i \tag{19}$$

Steady-state solutions of equations of the type considered in Problem A are the solutions of an important class of elliptic partial differential equations. Consider the following problem for this type of equation.

Problem B

Determine $\phi(r_0)$ such that:

$$(1) \quad L_{\bar{r}_0} \phi(\bar{r}_0) = 0 \tag{20}$$

is satisfied within a bounded region R ;

(2) a piecewise continuous boundary condition $\phi_c(\bar{r}_b)$ is satisfied on the boundary C of R , i.e.,

$$\phi(\bar{r}_b) = \phi_c(\bar{r}_b) \tag{21}$$

The subscript t_0 is deleted from operator $L_{\bar{r}_0, t_0}$ to indicate that $L_{\bar{r}_0}$ is independent of time.

The solution of Problem B is the solution of a corresponding problem of type A for $t_0 = -\infty$. Since Problem B is independent of time, the required expected value can be obtained by starting random walks at $t_0 = 0$ rather than $t_0 = -\infty$ and allowing each walk to continue until a boundary point is reached. The expected value of the boundary values at such terminal points is

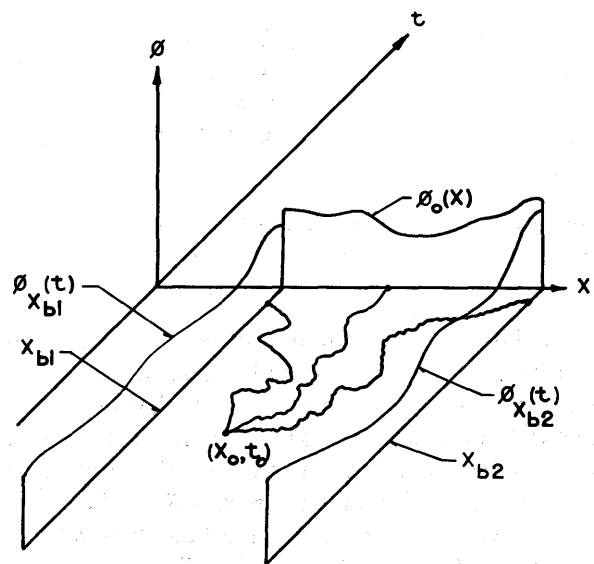


Figure 2. Random walks and initial and boundary conditions of a typical problem with one space variable.

$$\phi(\bar{r}_o) = \int_0^\infty \int_C \phi_c(\bar{r}_b) g(\bar{r}_b, t_b | \bar{r}_o, 0) dr_b dt_b \quad (22)$$

As was shown for Problem A, it follows the expected value of Eq. (22) is a solution of Problem B. An average of the same form as that of Eq. (19), in which the ϕ_i are boundary values selected by random walks starting at \bar{r}_o , approximates the expected value and is the Monte Carlo solution.

Solutions of Problems A and B can be combined to yield solutions of a class of nonhomogeneous partial differential equations as defined by Problem C.

Problem C

Determine $\phi(\bar{r}_o)$ such that:

$$(1) \quad L_{\bar{r}_o} \phi(\bar{r}_o) = -H(\bar{r}_o) \quad (23)$$

is satisfied within a bounded region R , where $H(\bar{r}_o)$ is a piecewise continuous function;

(2) a piecewise continuous boundary condition $\phi_c(\bar{r}_b)$ is satisfied on the boundary C of R ; i.e.,

$$\phi(\bar{r}_b) = \phi_c(\bar{r}_b) \quad (24)$$

To obtain the solution of Problem C, consider the time-independent boundary value problem of type B:

- (1) $L_{\bar{r}_o} \phi_1(\bar{r}_o) = 0$ within R ,
- (2) $\phi_1(\bar{r}_b) = \phi_c(\bar{r}_b)$ on the boundary C of R ; and

the time-dependent boundary value problem of type A:

- (1) $\frac{\partial}{\partial t_o} \phi_2(\bar{r}_o, t_o) = L_{r_o} \phi_2(\bar{r}_o, t_o)$ within R ,
- (2) $\phi_2(\bar{r}_o, 0) = H(\bar{r}_o)$
- (3) $\phi_2(\bar{r}_b, t_o) = 0$

The solution of Problem C is given in terms of the solutions of the two subproblems by

$$\begin{aligned} \phi(\bar{r}_o, t_o) = & \int_R \phi_o(\bar{r}_2) \exp - \int_{t_o}^0 d(\bar{r}(t), t) dt \quad \bar{r}(0) = \bar{r}_2 \quad f(\bar{r}_2, 0 | \bar{r}_o, t_o) dr_2 \\ & + \int_{t_o}^0 \int_C \phi_c(\bar{r}_b, t_b) \exp. - \int_{t_o}^{t_b} d(\bar{r}(t), t) dt \quad \bar{r}(t_b) = \bar{r}_b \quad g(\bar{r}_b, t_b | \bar{r}_o, t_o) dr_b dt_b \\ & \bar{r}(t_o) = \bar{r}_o \quad \bar{r}(t_o) = \bar{r}_o \end{aligned} \quad (28)$$

$$\phi(\bar{r}_o) = \phi_1(\bar{r}_o) + \int_{-\infty}^0 \phi_2(\bar{r}_o, t_o) dt_o \quad (25)$$

This solution is verified by substituting Eq. (25) into the conditions of Problem C and using the information provided by the subproblems.

The Monte Carlo solution of Problem C is obtained by determining $\phi_1(\bar{r}_o)$ and $\phi_2(\bar{r}_o, t_o)$ by the methods of Problems A and B and then integrating $\phi_2(\bar{r}_o, t_o)$ with respect to t_o by some numerical technique.

More general partial differential equations than those for which Monte Carlo methods have been outlined can be solved by using Eq. (12) and (13) for density functions u and v . From the definitions of u and v (Eqs. (9) and (10)) it follows that u and v satisfy the same initial and boundary conditions as density functions f and g respectively. It therefore follows that problems similar to Problems A, B, and C but with $L_{\bar{r}_o, t_o}$ and $L_{\bar{r}_o}$ replaced by $L_{\bar{r}_o, t_o} - d(\bar{r}_o, t_o)$ and $L_{\bar{r}_o} - d(\bar{r}_o)$ respectively have solutions that can be written as expected values with respect to functions u and v . For example,

$$-\frac{\partial \phi}{\partial t_o}(\bar{r}_o, t_o) = L_{\bar{r}_o, t_o} \phi(\bar{r}_o, t_o) - d(\bar{r}_o, t_o) \phi(\bar{r}_o, t_o) \quad (26)$$

with initial and boundary conditions as given by Problem A has solution

$$\begin{aligned} \phi(\bar{r}_o, t_o) = & \int_R \phi_o(\bar{r}_2) u(\bar{r}_2, 0 | \bar{r}_o, t_o) dr_2 \\ & + \int_{t_o}^0 \int_C \phi_c(\bar{r}_b, t_b) v(\bar{r}_b, t_b | \bar{r}_o, t_o) dr_b dt_b \end{aligned} \quad (27)$$

From definitions (9) and (10), Eq. (27) becomes

The expected value $\phi(\bar{r}_o, t_o)$ given by Eq. (28) can be approximated by the average $\phi_N(\bar{r}_o, t_o)$ of the product $\gamma_i \phi_i$ for walks originating at (\bar{r}_o, t_o) where ϕ_i is the initial or boundary value at the terminal point of the i th walk and γ_i is the value

$$\gamma = \exp - \int_{t_o}^{\tau} d(\bar{r}(t), t) dt \quad (29)$$

for the corresponding walk. The upper limit of integration, τ , is 0 for walks terminating at $t = 0$, and t_b for walks terminating at a boundary. The Monte Carlo solution is therefore

$$\phi_N(\bar{r}_o, t_o) = \frac{1}{N} \sum_{i=1}^N \gamma_i \phi_i \quad (30)$$

In a manner similar to above it follows that if L_r in Problem B is replaced by $L_r - d(\bar{r}_o)$ the Monte Carlo solution is as given by Eq. (30). In this case each ϕ_i is a boundary value selected by a random walk that starts at \bar{r}_o and each γ_i is a corresponding value of

$$\gamma = \exp - \int_{t_o=0}^{t_b} d(\bar{r}(t)) dt \quad (31)$$

The superposition of solutions of Problems A and B to obtain solutions of Problem C also applies to equations containing the $d(\bar{r}_o)$ term.

HYBRID COMPUTER MECHANIZATION OF MONTE CARLO METHODS

A computing system for mechanizing the Monte Carlo methods that have been developed must carry out the following operations.

- (A) Simulate stochastic differential Eqs. (1) to (3).
- (B) Evaluate γ as given by Eq. (29) or (31).

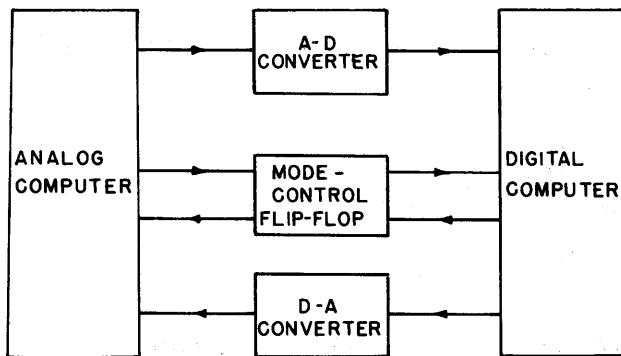


Figure 3. Hybrid computer system for Monte Carlo methods.

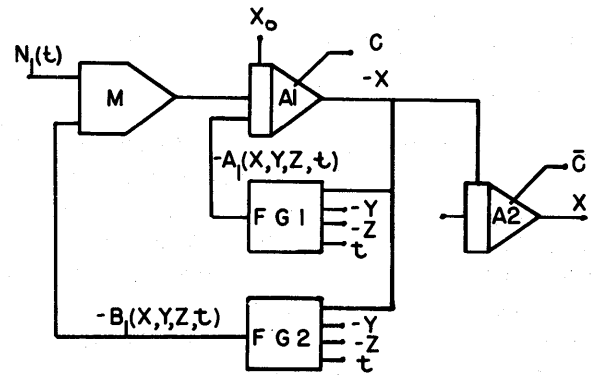


Figure 4. Block diagram for simulation of a stochastic differential equation.

- (C) Terminate solution of the stochastic differential equations at either time $t = 0$ or whenever a boundary is reached.
- (D) Generate the initial or boundary values corresponding to the terminal points of the random walks.
- (E) Form the averages given by Eq. (19) or (30).

Automatic readout of the solutions $\phi_N(\bar{r}_o, t_o)$ and adjustment of (\bar{r}_o, t_o) after each set of N random walks is also desirable.

A hybrid system in which synchronism of the two computers is realized by a mode-control flip-flop is shown in Fig. 3. Operations (A) to (D) listed above are performed by the analog computer whereas operation (E) as well as adjustment of the analog computer and readout of solutions are handled by the digital computer.

SIMULATION OF STOCHASTIC PROCESSES

An analog computer block diagram for simulating stochastic differential Eq. (1) in its most general form is shown in Fig. 4. The function generation indicated in this figure can be realized simply with diode function generators and multipliers whenever closed-form mathematical expressions are known for the functions or whenever the functions are of only a single variable. Special techniques⁷ are required for generation of functions of a more general form.

Integrator A2 shown in Fig. 4 is used to "track-and-hold" the random variable x . This track-and-hold feature is used to hold the terminal value of the random vector \bar{r} , and consequently the initial or boundary value ϕ_i generated from \bar{r} , at a constant value while ϕ_i or $(\gamma_i \phi_i)$ is read by the digital computer.

The mode-control signal c and its logical inverse \bar{c} synchronize the track-and-hold modes of integrator A2 with the compute and initial-condition modes respectively of integrator A1.

The uncorrelated Gaussian white noise terms N_i that drive the stochastic differential Eqs. (1) to (3) must be simulated on the analog computer by noise sources that are physically realizable. Noise sources with characteristics that approximate the ideal characteristics can be derived from gas tubes, pseudo-random noise generators⁸ or discrete-interval binary noise generators.⁹ The example solutions given in this paper were obtained with a discrete-interval binary noise generator that was multiplexed to give three essentially uncorrelated noise channels.

For problems in which γ must be generated, the following implicit method is used. From Eq. (29)

$$\begin{aligned} \frac{d\gamma}{dt} &= -d(\bar{r}(t), t)\gamma \\ \gamma(t_0) &= 1 \end{aligned} \quad (32)$$

The response γ of these equations is obtained easily on the analog computer. The mode of the integrator that is used to simulate Eq. (32) is controlled by the signal c of the mode-control flip-flop.

DETECTION OF BOUNDARIES

The mode-control flip-flop is set when a random walk reaches a boundary or at the terminal time $t = 0$ by using the outputs of suitably driven analog voltage comparators. The flip-flop is reset by the digital computer after the boundary or initial value

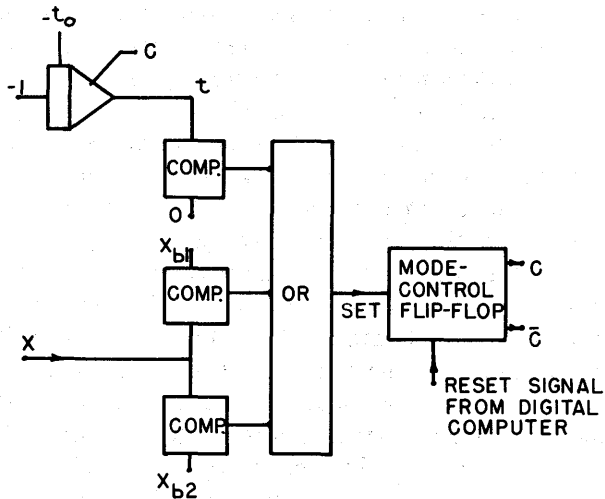


Figure 5. Boundary detection for problems with one space variable and one time variable.

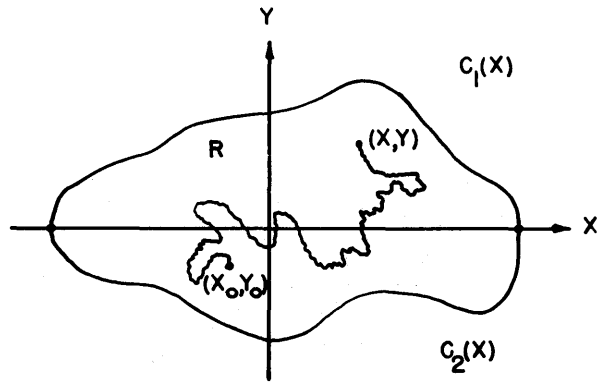


Figure 6. Two-dimensional region.

at the terminal point of the random walk has been read into the digital computer for averaging. For example, for the problem illustrated in Fig. 2 the scheme shown in Fig. 5 would be used to trigger the mode-control flip-flop.

The boundaries of problems with two or more spatial variables can be detected by using function generators together with voltage comparators. Consider the two-dimensional region R shown in Fig. 6, in which the curves $C_1(X)$ and $C_2(X)$ that form C are both single-valued functions of X . It is clear from the figure that a random walk with instantaneous components (x,y) reaches C whenever $y = C_1(x)$ or $y = C_2(x)$. These boundary detection criteria are implemented on the analog computer by the method shown in Fig. 7.

The boundaries of a region as shown in Fig. 8 in which $C_1(U)$ and $C_2(U)$ are single valued functions of position U along a dividing line D are detected by using a coordinate transformation. That is, variables x and y are transformed to u and v by the transformation

$$\begin{aligned} u &= x \cos\beta - y \sin\beta - a \\ v &= x \sin\beta + y \cos\beta - b \end{aligned} \quad (33)$$

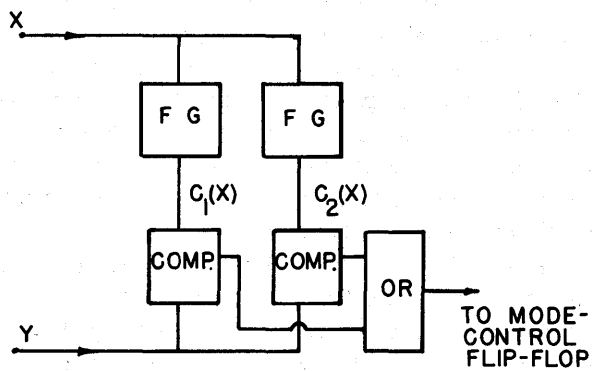


Figure 7. Two-dimensional boundary detection.

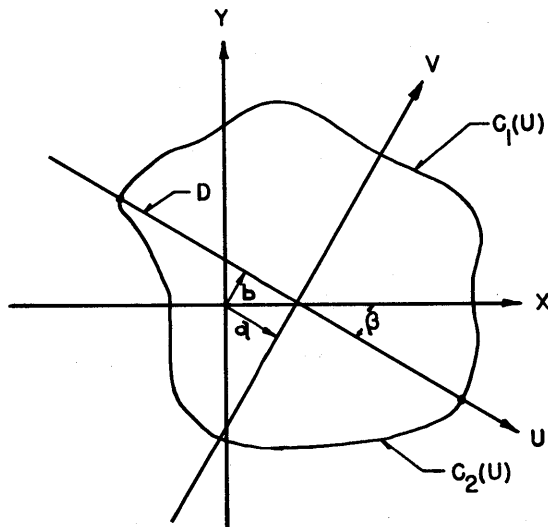


Figure 8. Coordinate transformation used for boundary detection of two-dimensional regions.

In u, v coordinates the criteria that a random walk is at the boundary is $v = C_1(u)$ or $v = C_2(u)$.

The boundaries of simply connected regions of arbitrary shape are detected by dividing the region into simple regions R_1, R_2, \dots, R_n that have dividing lines D_i . The exit of a random walk from each simple region is detected by the previous methods. The resultant signals are combined with an AND gate to give a signal when the walk leaves all R_i and hence the total region R .

The boundaries of circles and ellipses of the form

$$\left(\frac{X+a}{b}\right)^2 + \left(\frac{Y+c}{d}\right)^2 = 1$$

are detected by comparing the function $f(x,y) = \left(\frac{x+a}{b}\right)^2 + \left(\frac{y+c}{d}\right)^2$

with 1. Hence only two multipliers and one comparator are required for these common regions.

The preceding methods can be generalized to three-dimensional regions by using a dividing plane separating the boundary into surfaces that are single-valued functions of position on the plane. For boundary surfaces which are simple functions of the two plane variables, the method is easy to apply. If this is not the case, special purpose function generator techniques⁷ are necessary.

The boundaries of three-dimensional regions with some type of symmetry can often be detected by combining the methods described for one- and two-dimensional regions. For example, cubic regions are detected by using three pairs of comparators in the

same manner that a single pair is used for one-dimensional problems. In addition, the boundaries of spheres and ellipsoids can be detected with three multipliers and a single comparator in the same manner that two multipliers and a comparator are used for circles and ellipses.

GENERATION AND AVERAGING OF INITIAL AND BOUNDARY VALUES

At the instant a boundary C is reached, the mode-control flip-flop is triggered from a comparator by the methods that have been discussed. The triggering of this flip-flop places track-and-hold amplifiers in the hold mode so that the terminal values of the components x, y , and z of \bar{r} are available as constant voltages on the analog computer. The initial and boundary values ϕ_i are generated with function generators from these components of \bar{r} .

The function generation prescribed above can be carried out with function generators and multipliers whenever the boundary values are known as simple functions of x, y and z or whenever they can be expressed as a function of a single variable. For two-dimensional problems in which a dividing line D is used for detecting the boundaries, the boundary values are conveniently generated as a function of the variable u defined along the dividing line.

When the values ϕ_i cannot be generated conveniently by analog computer techniques, they can always be generated within the digital computer. When the digital computer is used for function generation, the components x, y and z of the terminal position vector are read; then a table stored within the computer is scanned, or some other method is used, to determine the corresponding value of ϕ_i . Since more than one value must be read by the digital computer and since additional digital operations are required, this procedure with slow digital equipment is more time-consuming than analog function generation. However, with fast digital equipment it is possible to store the terminal components x, y and z of the i th walk with a track-and-hold arrangement and read them during the $(i+1)$ st walk. Thus, if the conversion equipment and digital computer are sufficiently fast, the values x, y and z can be read and the digital function generation for the i th walk can be carried out while the analog computer is simulating the $(i+1)$ st random walk. This procedure is very efficient in that essentially no time is wasted between walks.

The average $\phi_N(\bar{r}_o, t_o)$ for each point (\bar{r}_o, t_o) at which a solution is desired is formed by adding, in sequence, each value ϕ_i (or $\gamma_i \phi_i$) to a partial sum stored within the digital computer. A tally of the number of random walks that have been completed is kept by the digital computer and after N walks the average is typed out. A digital computer, or at least a digital method, is necessary for the averaging operation because for N large (1000-40,000) a large dynamic range is required to obtain the sum precisely. After the solution has been obtained at a point the digital computer adjusts the starting point (\bar{r}_o, t_o) on the analog computer so that the solution at another point can be obtained. In this manner the solutions at all points of interest are obtained.

EXAMPLES

The examples given in this paper were selected from a large number of problems that were solved using an EAI 231 R-V analog computer and a Logistics Research Alwac III-E digital computer. With these slow computers, it was possible to simulate about 10 random walks per second. To obtain solutions with a small variance, 1000 random walks were used for each point solution.

Solutions of a one-dimensional time-independent problem for three parameter values are shown in

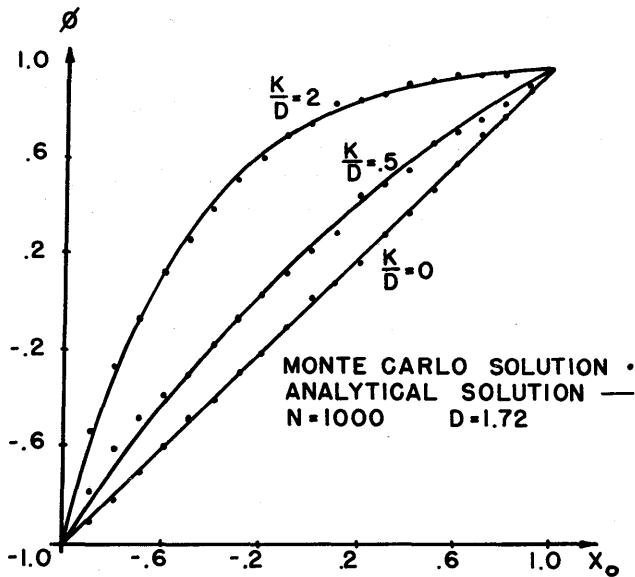


Figure 9. Solutions of $D_1 \frac{d^2\phi}{dx_o^2} + K \frac{d\phi}{dx} = 0$, for $\phi(-1) = -1$ and $\phi(1) = 1$.

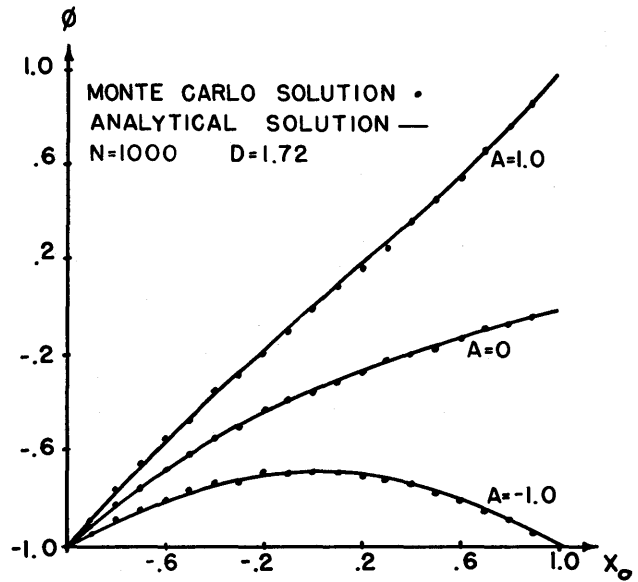


Figure 10. Solutions of $\frac{d^2\phi}{dx_o^2} - (1 - x_o^2)\phi = 0$, for $\phi(-1) = -1$ and $\phi(1) = A$.

Fig. 9. When $K/D_1 = 0$, the average duration of a random walk for this problem is

$$T(x_o) = \frac{1 - x_o^2}{2D_1} \tag{34}$$

Since $T(x_o)$ is very easily measured, Eq. (34) provides a simple method for determining the power spectral density $2D_1$ of the noise source.

The solutions of another one-dimensional problem for three different boundary values are shown in Fig. 10. This problem is of the type that requires generation of the functional γ . According to Eq. (31) the required functional is

$$\gamma = \exp - D_1 \int_{t_o=0}^{t_b} (1 - x^2) dt \tag{35}$$

As a final example, solutions of the diffusion equation

$$-\frac{\partial\phi}{\partial t_o}(\bar{r}_o, t_o) = \nabla^2\phi(\bar{r}_o, t_o) \tag{36}$$

with initial condition $\phi_o(\bar{r}_o) = -1$ and boundary condition $\phi_c(\bar{r}_b) = 1$ are shown in Fig. 11. The solutions shown are for the center of a line, square and cubic region. From this example, it is significant to note that the average time $T(0)$ for a random walk to reach a boundary decreases with the dimension of the problem.

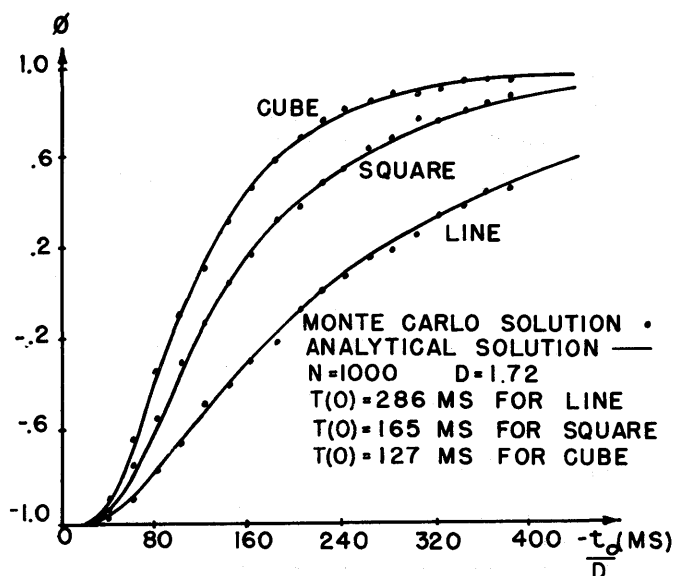


Figure 11. Solutions of the diffusion equation at the center of a line, a square and a cubic region.

CONCLUSIONS

Monte Carlo methods have been developed for obtaining approximate solutions to partial differential equations of a very general form. The methods are easily mechanized with a small analog computer coupled by means of a modest interface to a small digital computer.

The Monte Carlo solutions are obtained sequentially in a point by point manner. Therefore, if solutions at only a few points are required the methods may be more efficient, even using slow computers, than conventional finite-difference methods. On fast computers, and especially special-purpose fast computers,¹⁰ the methods should provide a powerful means for solving many types of partial differential equations.

REFERENCES

1. G. E. Forsythe and W. R. Wasow, *Finite-Difference Methods for Partial Differential Equations*, Wiley & Sons, New York, 1960.
2. T. J. H. Curtiss, "Sampling Methods Applied to Differential and Difference Equations," *Proc. Seminar on Scientific Computation*, International Business Machines Corp., New York, Nov. 1949.
3. K. Chuang, L. F. Kazda and T. Windeknecht, "A Stochastic Method of Solving Partial Differential Equations using an Electronic Analog Computer," Project Michigan Report 2900-91-T, Willow Run Laboratories, University of Michigan (June 1960).
4. W. D. Little, "Hybrid Computer Solutions of Partial Differential Equations by Monte Carlo Methods," PhD thesis, University of British Columbia, Oct. 1965.
5. A. T. Bharucha-Reid, *Elements of the Theory of Markov Processes and their Applications*, McGraw-Hill, New York, 1960, Chap. 3.
6. D. A. Darling and A. J. Siegert, "A Systematic Approach to a Class of Problems in the Theory of Noise and Other Random Phenomena, Part I," *IRE Trans. on Information Theory*, vol. IT-3, pp. 32-37 (Mar. 1957).
7. R. H. Wilkinson, "A Method of Generating Functions of Several Variables using Analog Diode Logic," *IEEE Trans. on Electronic Computers*, vol. EC-12, pp. 112-29 (Apr. 1963).
8. R. L. T. Hampton, "A Hybrid Analog-Digital Pseudo Random Noise Generator," *Simulation*, vol. 4, no. 3, pp. 179-85 (Mar. 1965).
9. H. Kohne, W. D. Little and A. C. Soudack, "An Economical Multichannel Noise Source," *ibid.*, vol. 5, no. 8 (Nov. 1965).
10. G. A. Korn, "Hybrid Computer Monte Carlo Techniques," *ibid.*, vol. 5, no. 4, p. 234 (Oct. 1965).

PARAMETER OPTIMIZATION BY RANDOM SEARCH USING HYBRID COMPUTER TECHNIQUES *

G. A. Bekey

University of Southern California, Los Angeles, California

and

M. H. Gran, A. E. Sabroff, A. Wong

TRW Systems, Redondo Beach, California

INTRODUCTION

Optimum selection of the parameter values for a complex dynamic system usually consists of three distinct phases: (1) a proposed system configuration is selected, in which only parameter values remain as unknowns; (2) one or more performance or cost criteria for evaluation of the system are selected; and (3) a computer technique or algorithm is chosen for adjusting the system parameters until an optimum value of the criterion function is achieved. Typical algorithms are those based on relaxation or steep descent methods.¹ However, both of these methods are primarily suited to optimization of criterion functions with unique minima or maxima. Furthermore, they may fail to converge or may converge only very slowly if the criterion function—parameter space exhibits “ridges”² or if the criterion function is only piecewise differentiable or piecewise continuous. Both of these difficulties are likely to arise in con-

nection with nonlinear systems. This paper presents an approach to finding a global optimum by means of a modified sequential random perturbation technique implemented on a hybrid computer.

Random search techniques for parameter optimization were originally proposed by Brooks.³ They were successfully implemented on analog computers by Munson and Rubin⁴ and Favreau and Franks.⁵ A hybrid computer implementation using only two parameters, a fixed step size, and an algebraic criterion function was studied by Mitchell.^{6,7} This paper extends the previous work by applying it to a nonlinear dynamic system with nine parameters. Furthermore, the effects of initial conditions and analog computer errors are included in the optimization program in a systematic way. The method is illustrated by application to the optimization of a satellite acquisition system.⁸

PROBLEM FORMULATION

The dynamical system to be optimized is described by the differential equation:

$$\dot{\underline{x}} = \underline{f}(\underline{x}; t; \underline{\alpha}) \quad (1)$$

* The work reported in this paper was supported by the U.S. Air Force Flight Dynamics Laboratory, Research and Technology Division, Air Force Systems Command, Wright Patterson Air Force Base, Ohio, under Contract No. AF 33(615)-135.

where \underline{x} is the $(n \times 1)$ state vector, f is an $(n \times 1)$ vector function, $\underline{\alpha}$ is an $(m \times 1)$ parameter vector and t represents running time. It is desired to study this dynamic system over a large class of initial conditions. Define \underline{X}_0 as the set of all $(n \times 1)$ initial state vectors which are of interest to be studied and an element of \underline{X}_0 as \underline{x}_0 . A unique solution of (1) is solely dependent on \underline{x}_0 , $\underline{\alpha}$ and t and therefore can be represented as:

$$\underline{x} = \underline{x}(\underline{x}_0; t; \underline{\alpha}) \quad (2)$$

A cost or criterion function can be written ordering the desirability of the particular choice of $\underline{\alpha}$ for a given \underline{x}_0 as:

$$J(\underline{x}_0; \underline{\alpha}) = \int_0^t g(\underline{x}; t; \underline{\alpha}) dt \quad (3)$$

where g is a scalar function of \underline{x} , $\underline{\alpha}$ and t . Examples of J may be fuel consumed or time required for satellite acquisition for a given parameter and initial condition set.

For a given initial condition and parameter setting, equation (3) provides a scalar value describing the "quality" of the dynamic system in a quantitative fashion. As it is desired to study the effect of the parameter settings over the entire space of initial conditions, a new criterion function must be defined as:

$$F(\underline{\alpha}) = \int_{\substack{\text{all} \\ \underline{X}_0}} h[J(\underline{x}_0; \underline{\alpha})] d\underline{X}_0 \quad (4)$$

where h is a scalar function of the functional $f(\underline{x}_0; \underline{\alpha})$ and the integral is a Riemann-Stieltjes integral which allows integration of discrete \underline{X}_0 spaces as well as continuous. This provides a single measure of the "quality" of a selection $\underline{\alpha}$ over the entire space of initial conditions. Examples of F studied in this paper are:

$$F_1 = \max_{\underline{x}_0 \in \underline{X}_0} f(\underline{x}_0, \underline{\alpha}) \quad (5a)$$

$$F_2 = \sum_{j=1}^q f(\underline{x}_{0j}, \underline{\alpha}) \quad (5b)$$

$$\underline{x}_0 \in \underline{X}_0$$

$$F_3 = \sum_{j=1}^q \max_{\underline{x}_{0j} \in \underline{X}_j} f(\underline{x}_{0j}, \underline{\alpha}) \quad (5c)$$

For a given criterion functional F , a computer algorithm is desired which finds the optimum $\underline{\alpha}$ which will be denoted by $\underline{\alpha}^*$,

$$F^*(\underline{\alpha}^*) = \min_{\underline{\alpha}} F(\underline{\alpha}) \quad (6)$$

Values of $\underline{\alpha}^*$ for different criteria will, in general, be different. With nonlinear differential equations, the criterion functions cannot be assumed to possess a unique minimum or maximum, and the optimization algorithms must be designed to seek the global extremum. Since for any physical problem, the allowable range of all parameters is limited, admissible parameter vectors will be constrained to the allowable region of the parameter space. Local continuity of the criterion function will be assumed.

Exhaustive search for an optimum design, in which parameter values are quantized and all possible parameter combinations are tested, is clearly limited to systems with only a few parameters. Consequently, an organized search procedure is required.

Let the initial parameter choice be indicated by $\underline{\alpha}^{(0)}$, so that the initial criterion function is

$$F_0 = F(\underline{\alpha}^{(0)}) \quad (7)$$

If the parameter adjustment is made according to the gradient method, then a parameter increment is computed from

$$\Delta \underline{\alpha}^{(0)} = k \nabla F(\underline{\alpha}^{(0)}) \quad (8)$$

where k is a scaling matrix.

Gradient methods suffer from three major disadvantages for systems of the type being considered here. First, the computation of the gradient requires m trial steps to determine each component; second, the method leads only to a local minimum of the criterion function, and third, the gradient method encounters significant difficulties if the criterion function exhibits "ridges" or "narrow valleys" in the parameter space.²

AN ALGORITHM FOR RANDOM SEARCH OPTIMIZATION

Strictly speaking, pure random search refers to a computation of the criterion function at a number of randomly chosen points in the parameter space, and selection of the particular parameter values (α_1, α_2) yielding the smallest value of $F(\alpha)$. However, such a sequence of randomly selected parameter vectors does not take advantage of the local continuity

properties of most criterion function surfaces. Consequently, the strategy to be discussed below should more properly be referred to as "sequential random scanning" or "random creep".⁷ Assume that the initial parameter vector is again designated by $\underline{\alpha}^{(0)}$. Now, choose an increment $\underline{\Delta\alpha}^{(0)}$ by selecting the individual parameters $\Delta\alpha_i^{(0)}$, $i = 1, 2, \dots, m$ from m Gaussian sequences of random numbers with mean zero and variance c_i . Then, if the m random sequences are independent, the orientation and length of the parameter increment $\underline{\Delta\alpha}^{(0)}$ will be random, and a trial value

$$\underline{\alpha}' = \underline{\alpha}^{(0)} + \underline{\Delta\alpha}^{(0)} \quad (9)$$

is obtained. The criterion function $F' = F'(\underline{\alpha}')$ is computed and compared to $F_0 = F(\underline{\alpha}^{(0)})$. If there is an improvement, the parameters are updated by letting $\underline{\alpha}' = \underline{\alpha}^{(1)}$. If there is no improvement, the trial step is abandoned and a new trial step is chosen. This basic strategy is illustrated in the flow chart of Fig. 1.

Now from the standpoint of computer implementation, the differential equations can be solved on the analog computer for each trial value $\underline{\alpha}'$. The random increments $\underline{\Delta\alpha}_i$ can be obtained by sampling analog noise generators, by generating pseudo-random sequences in the digital computer, or by construction of special devices such as shift register noise generators with several independent outputs.⁷

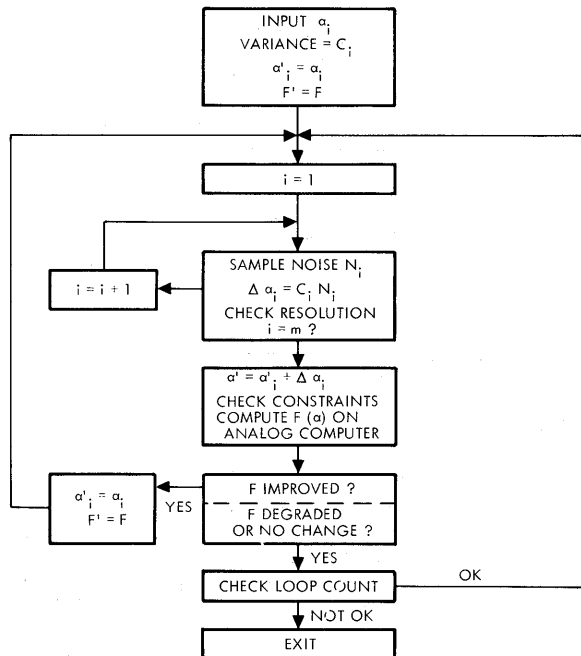


Figure 1. The basic algorithm.

Modifications of the Basic Algorithm

In order to take maximum advantage of the properties of the criterion surface, the basic strategy can be modified in a number of ways. Successive steps can be made correlated in such a way as to favor successful direction. The mean of the distribution of steps can be biased in the direction of a successful step, or after a specified number of successive successes.⁶ Thus, the j th trial step could be computed from

$$\underline{\Delta\alpha}^{(j)} = C^{(j)} \underline{N}^{(j)} + \underline{b}^{(j)} \quad (10)$$

where $\underline{N}^{(j)}$ is a column vector of Gaussian random samples, $C^{(j)}$ is a diagonal matrix of variances, and $\underline{b}^{(j)}$ is a bias vector, which may be altered after one or more successes (or failures).

In the present study, the term "absolute biasing" has been used to denote the repeated use of a successful random step as long as continued success is attained. That is, if $\underline{\Delta\alpha}^{(j)}$ is successful, we choose

$$\underline{\Delta\alpha}^{(j+1)'} = \underline{\Delta\alpha}^{(j)} \quad (11)$$

and test for success. If $\underline{\Delta\alpha}^{(j)}$ was a failure, one chooses

$$\underline{\Delta\alpha}^{(j+1)'} = -\underline{\Delta\alpha}^{(j)} \quad (12)$$

Such a technique will be referred to as "absolute positive and negative directional biasing."

It is also possible to adjust the variance of the distribution of step sizes. For example, as a local minimum is approached, the variance can be decreased in order to decrease the probability of overshooting the optimum.

The basic algorithm logically divides into two parts, concerned with the search for a local minimum and the global minimum respectively.

Search for a Local Minimum

This strategy, using absolute positive and negative biasing, is illustrated in Fig. 2. It consists of the following steps for the computation of the j th trial step:

1. A Gaussian random vector $\underline{N}^{(j)}$ is obtained. In the present study the components $\underline{N}_i^{(j)}$, $i = 1, 2, \dots, m$, were obtained from successive trial samples of an analog high-frequency noise generator. The sampling frequency was sufficiently low compared to the noise

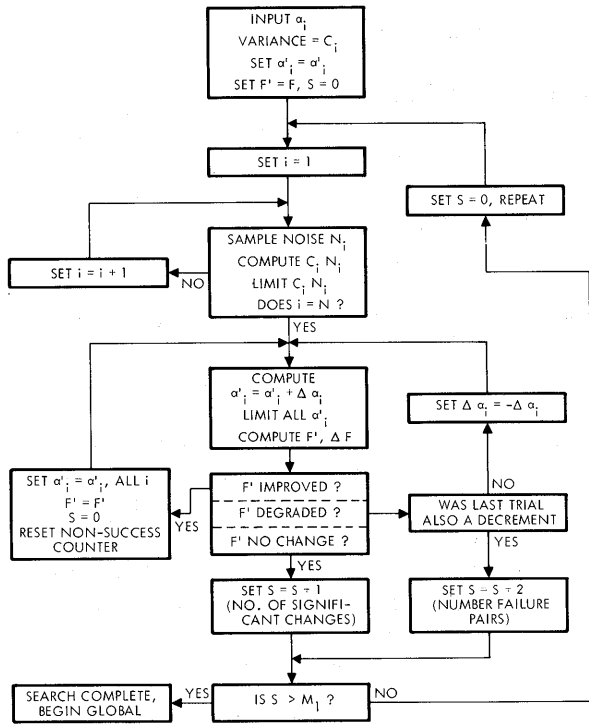


Figure 2. Absolutely biased local random search flow diagram.

bandwidth to insure that successive samples are essentially uncorrelated.

2. A trial step is computed with magnitude constraints imposed on all parameters.
3. The analog computer is used to compute a new value of $F = F'$.
4. F' is compared with $F^{(j)}$. If $F' < F^{(j)}$, we let $F' = F^{(j+1)}$ and $\alpha' = \alpha^{(j+1)}$. If $F' > F^{(j)}$, the increment $-\Delta\alpha^{(j)}$ is tried.
5. The number of trials which leads to either no change or an increase in F is counted and used as a stopping criterion.

Search for the Global Optimum

Once a local optimum has been found, it must be tested to determine whether it is indeed the global optimum. The approach selected in this study is a random search. This allows much flexibility in that the statistics of the search may be adjusted to correspond to estimates of the location of other likely optima. For the present case, it was felt the space near the local optimum was the most likely location

for an even better optimization criterion. The strategy then consisted of randomly sampling numbers corresponding to the parameters (as in the local random search), starting with a very small variance and expanding this variance slowly if no better points are found. If an improvement is found, a local search strategy is once again initiated. Figure 3 gives the flow detail of the global random search algorithm.

Initial Condition Set Selection

Equation (3) suggests that natural criterion functions such as fuel consumption or acquisition time are functions of the initial conditions as well as parameter values. If both the differential equation and criterion function (J) were linear, a single set of parameters could be found which would optimize the J for all initial conditions simultaneously. The present study is concerned with the more general case where both the differential equations and the criterion function may be nonlinear. In this case, the same set α may not optimize J for various initial conditions. In order to provide a single criterion function, the J 's must be synthesized in some fashion as indicated by Eq. (4). As the allowable I.C. space X_0 is generally a continuous set, an infinite number of I.C.'s exist. Apparently not all I.C.'s can be considered. The twofold problem then exists: (1) which

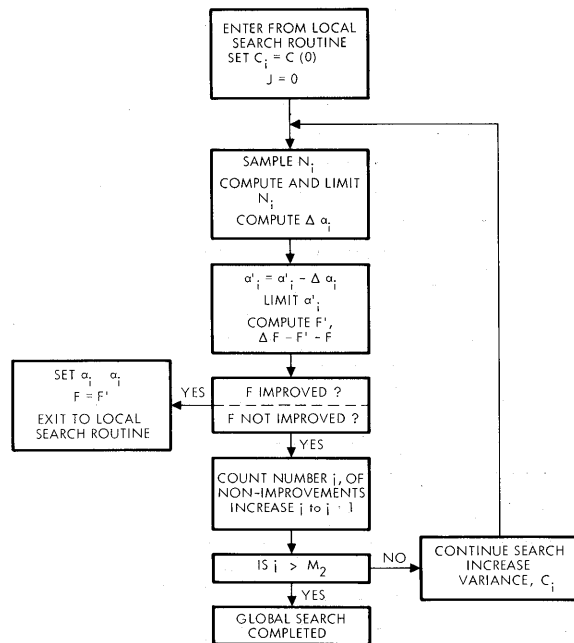


Figure 3. Random global search flow diagram.

initial conditions should be considered, and (2) how the synthesis should be accomplished.

If the criterion space is reasonably smooth it can best be described by partitioning the space and inspecting an initial condition from each partition. For the case of six variables (and, therefore, six initial conditions), as may exist for the satellite dynamic and kinematic equations, quantizing each variable into as few as five values yields $5^6 = 15,625$ different initial conditions. This increased cost due to high dimensions has been aptly described by Bellman as the "curse of dimensionality."⁹

The synthesis function h may serve to reduce the dimensionality. Frequently all that is desired is a reasonable set of initial conditions which will describe the worst possible conditions for F .

This is the minimax criterion:

$$F^*(x^*) = \min_{\alpha} \max_{x_0 \in X_0} J(x_0, \alpha) \quad (13)$$

In some cases, the general location of this worst case can be estimated eliminating much of the initial condition space. The same algorithm employed to minimize J with respect to α can be used to maximize J with respect to the initial conditions. This approach has been programmed but the results are not included in this paper. The minimax criterion may unfairly penalize the more likely cases. For this reason, the synthesis procedure used in this paper was the most obvious linear relation—an unweighted average:

$$F = \sum_{j=1}^q J(x_{0j}, \alpha) \quad (14)$$

The most desirable aspects of Eqs. (13) and (14) can be combined by the following procedure. The I.C. space is divided into q subdivisions which we designate by X_1, \dots, X_q . For a given α , each of these subspaces are searched for the maximum initial condition set. These maximum I.C. points are then algebraically summed as in Eq. (14). Equation (15) expresses the operation:

$$F = \sum_{j=1}^q J^*(x_{0j}, \alpha) \quad (15)$$

where $x_{0j} \in X_j$ and

$$J^*(x_{0j}, \alpha) = \max_{x_{0j} \in X_j} J(x_{0j}, \alpha)$$

As this type of search procedure for every trial α is prohibitively expensive in computer time, the

maximization is only performed after a given number of improved α 's have been found. Experience with the acquisition simulation indicates these worst I.C.'s remain nearly the worst case for a large variation in α .

A SATELLITE ACQUISITION OPTIMIZATION

The general acquisition problem considered is that of aligning a single axis of a satellite parallel to a desired vector and driving the angular rotation about this axis to zero (one-axis acquisition problem). The reference coordinate frame is a three-dimensional, Cartesian set $(\hat{i}_1, \hat{i}_2, \hat{i}_3)$ with the three axes defining the desired pointing direction. The kinematic representation consists of the three direction cosines of the satellite axis to be aligned with the three reference axes. The control system can consist of any collection of sensors (that can be described by the six variables), whose outputs are processed by a compensation network which can be described by the parameters to be optimized. The outputs of the networks are then used to drive angular acceleration devices (torquers). In order to provide a more concrete control equation, the sensors are assumed to be three rate sensors and two direction cosine sensors. The control laws were chosen to be proportional but saturable control. The equations expressing the acquisition dynamics, kinematics, control law, several potential optimization criteria and the end of run criterion are given in Table 1. Six initial conditions (three attitudes and three rates) were specified for each trial optimization.

Analog Simulation

The above-mentioned equations were simulated on a Beckman 2132 analog computer. The end of run criterion indicated in Table 1 is necessary to determine when acquisition is complete. This is especially critical when the time of acquisition is the optimization criterion. The criterion selected is the absolute value of a weighted sum of the variables biased by some fixed voltage (see "Computer Error Detection" below). The criterion is assumed satisfied when this sum becomes zero. The bias is necessary as noises and drifts by the analog would prevent an unbiased zero from ever occurring. The bias level selected is more a function of actual voltage levels than their equivalent variable units.

Table 1. Simulated Acquisition Equations

Item Description	Equations	Discussion
Kinematics	$\dot{a}_{13} = \omega_2 a_{33} + \omega_3 a_{23}$ $\dot{a}_{23} = \omega_1 a_{33} - \omega_3 a_{13}$ $\dot{a}_{33} = \omega_2 a_{13} - \omega_1 a_{23}$	a_{i3} —Direction cosine from i body axis to 3 inertial ω_i —Body rate about i body axis
Dynamics	$\dot{\omega}_1 = [(I_2 - I_3)/I_1] \omega_2 \omega_3 + \gamma_1$ $\dot{\omega}_2 = [(I_3 - I_1)/I_1] \omega_1 \omega_3 + \gamma_2$ $\dot{\omega}_3 = [(I_1 - I_2)/I_3] \omega_1 \omega_2 + \gamma_3$	I_i —Inertia about i principal inertia axis γ_i —Control angular acceleration about i axis
Control laws	$\gamma_1 = -\alpha_1 (a_{23} + \alpha_3 \omega_1)$ $\gamma_2 = -\alpha_2 (-a_{13} + \alpha_4 \omega_2)$ $\gamma_3 = -\alpha_5 \omega_3$	α_i ($i = 1, 2, 3, 4, 5$)—Control law gain constants
End-of-run criterion	$\sum_{i=1}^3 b_i \omega_i + \sum_{i=1}^2 c_i a_{i3} - K = 0$	b_i, c_i —Weighting and scaling constants K —a constant
Minimum fuel criterion	$F = \int_0^T \sum_i \gamma_i dt$	
Minimum time criterion	$F = \int_0^T dt$	
Constraints	$\alpha_7, \alpha_9, \alpha_6$ α_8, α_{10} $\alpha_{11}, \alpha_{12}, \alpha_{13}$	Limits on magnitude of $\gamma_1, \gamma_2, \gamma_3$, respectively Saturation values of a_{13} and a_{23} , respectively Saturation values of $\omega_1, \omega_2, \omega_3$, respectively

Digital-Analog Interface

The digital computer provides those functions it is best suited for. It provides the optimization algorithm, initialization search procedure, controls the analog modes (Initial Condition, Operate), provides voltages for the initial conditions of the variables and parameter settings, and reads the optimization criterion from the analog.

In order to better understand the operations of the various elements of the optimization process a logical flow diagram of the entire computer operations in the optimization phase is shown in Fig. 4. An optimization criterion is selected (for example, minimum fuel or time). A set of initial conditions for the variables is selected based on the space search that has already been run. A best initial guess for the parameters is made. The error criterion (F) for this initial setting will be assumed already calculated.

A new point in parameter space is selected by the optimization algorithm. For each set of variable initial conditions, three analog computer runs are made giving a J for each run. The three J 's are compared to check parity (the reasons are explained later). If their standard deviation is within tolerance, the mean J for the three is selected as the criterion for this set of I.C.'s. This is the innermost loop (loop I). This process is repeated for each set of I.C.'s (loop II). The J 's for each I.C. set is averaged to give F . This is the F used to determine (loop III) whether the explored point is better than the present point. Loop III minimizes time or fuel as the error criterion.

COMPUTER ERROR DETECTION

The speed and accuracy of convergence of the optimization procedure is directly related to the

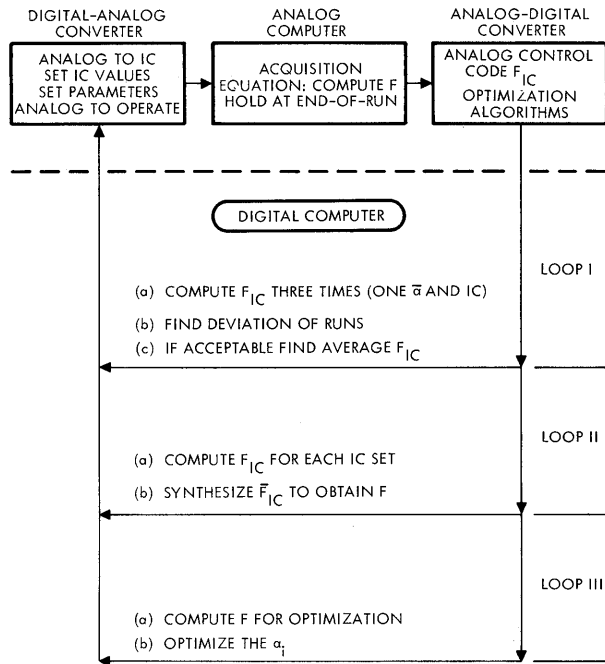


Figure 4. Optimization master logic flow diagram.

accuracy and repeatability of the analog simulation. Extensive testing of the analog computer found that for many runs in succession or even on separate days with identical inputs, the optimization criterion would be repeatable to within one volt unless an obvious malfunction occurred. The prime sources of malfunctions were: (1) a bit lost in the analog to digital converter, (2) a momentarily defective electronic switch, and (3) drift in the electronic multipliers. Most malfunctions were of a momentary nature so that for over 99% of the time no more than two successive runs were adversely affected. In light of this knowledge, the computer was programmed in the following manner to prevent computer malfunctions from negating an algorithm or consuming excessive time in trouble shooting.

Overload Detection

An overload indicates either an unstable differential equation or an equipment malfunction. The computer was programmed to halt on overload.

Time Limit Detection

Certain parameter combinations may result in extremely long optimization times. Occasionally, however, a computer malfunction could have the same effect. The computer was programmed to stop on a maximum time and try the problem again. If time is

exceeded again, it is printed out, a large value is assigned to the optimization criterion and the optimization continues automatically.

Standard Deviation Tests

The use of confidence tests provides a powerful tool for improving the accuracy of analog computer studies. Assume that errors in analog computer results are normally distributed with zero mean and a known standard deviation σ . The assumption of zero mean can be justified if the computer is balanced frequently and σ can be estimated from the sample variance V_s . Then, confidence tests can be used to determine the number of runs needed to satisfy a particular accuracy criterion. For example, suppose it is desired to be confident with .95 probability that the computer has no more noise than when σ was estimated. For $N = 3$ runs, we require that $\sqrt{V_s} \leq 3.0 \sigma$. When the allowable variance is exceeded, it is concluded that the computer is not operating properly. The computer makes three runs with the same inputs. If the standard deviation is less than one volt, operation is assumed normal and the optimization continues. If the standard deviation is greater than a volt, malfunction is assumed. In this case, the large standard deviation is printed out and another set of three runs attempted. If this set is accepted, the first set is discarded and the optimization is continued. The operator may stop the computation if he desires.

Note that if a set of runs is accepted, the three values of F are averaged, thus increasing the accuracy of the optimization criterion by $\sqrt{3}$.

RESULTS OF THE OPTIMIZATION STUDY

The specific numerical values used in the optimization are given in Table 2. A maximum of nine independent parameters were studied at one time. Fig. 5 shows a typical optimization using acquisition time as the criterion for the absolutely biased local random search. The nine parameter settings are plotted versus real time. The vertical lines indicate when the computer is in reset. Three trials are simulated and compared before the case is accepted as discussed in the preceding section. The optimization runs shown considered just one initial condition set. To indicate the improvements in system performance during the optimization for

Table 2. Numerical Values for Acquisition Optimization Study

Parameter	Description	Assumed Value or Range of Values	Units
ω_{im}	Maximum initial rate	± 70	milliradians/sec
ω_{ia}	Average initial rate	± 26	milliradians/sec
a_{ijn}	Nominal direction cosines	$a_{13} = a_{23} = 0 \ a_{33} = +1$	—
a_{ijmin}	Minimum initial attitudes	$\pm 1^a$	—
a_{ijmax}	Maximum initial attitudes	$+1, \pm 1/\sqrt{3}^a$	—
$(I_2 - I_3)/I_1$	Inertia ratios	+0.311	—
$(I_3 - I_1)/I_2$	Inertia ratios	-0.802	—
$(I_1 - I_2)/I_3$	Inertia ratios	+0.643	—
α_1	Roll position gain	0.001 to 0.100	sec ⁻²
α_2	Pitch position gain	0.001 to 0.100	sec ⁻²
α_3	Roll rate to position gain	0.001 to 0.100	sec ⁻²
α_4	Pitch rate to position gain	0.100 to 10.0	sec
α_5	Yaw rate gain	0.100 to 10.0	sec
α_6	Yaw torque limit	0.001 to 0.100	rad/sec ²
α_7	Roll torque limit	0.001 to 0.100	rad/sec ²
α_8	Pitch direction cosine limit	0.01 to 1.00	—
α_9	Pitch torque limit	0.001 to 0.100	rad/sec ²
α_{10}	Roll direction cosine limit	0.01 to 1.00	—

^a All physically realizable combinations.

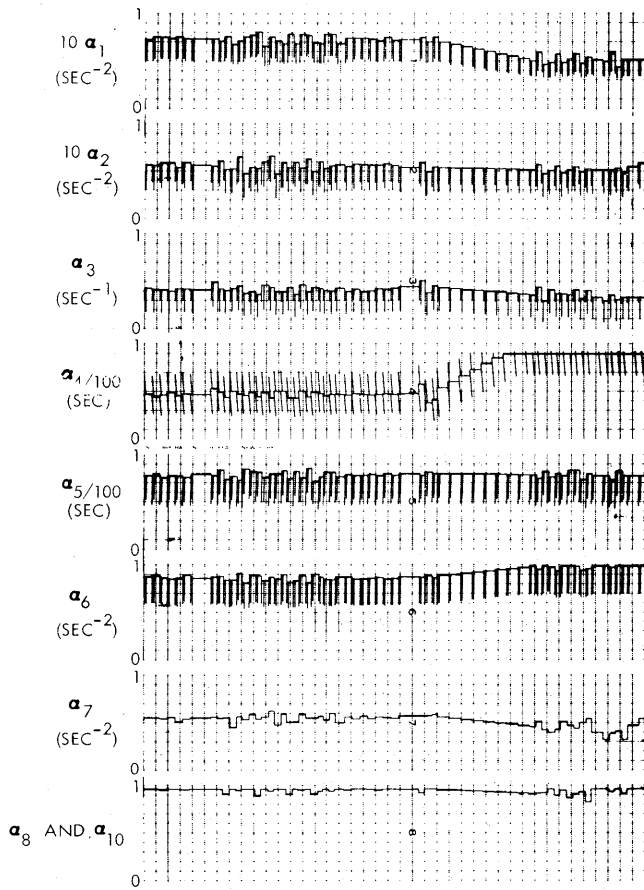


Figure 5. Nine parameter minimum time optimization using absolutely biased local random search.

one I.C. set, Fig. 6 gives the response of five independent variables ($\omega_x, \omega_y, \omega_z, a_{13}, a_{23}$) as defined in Table 1, the end-of-run criterion, time of run and fuel consumption for (1) the initial parameter settings, (2) the optimized minimum time settings, and (3) the optimized minimum fuel settings.

The major results of the study were the following:

1. Absolute biasing is an efficient way of improving the convergence of a random search process.
2. A successful strategy for changing the variance of the distribution of steps during the local search was not found. A subsequent study of USC¹⁰ verified the conclusion that a uniform variance yielded convergence to a local optimum as rapidly as any variance adjustment strategy attempted. In this study, a variance equal to 4% of the range between maximum and minimum limits on each parameter was used.
3. The random global search technique proved to be very useful. The strategy that appears most useful is that of using the local optimum as the origin and initiating the purely random (no biases) search with a very small variance. After each search, the variance is widened until any point in the parameter space has some chance of being inspected. For the most successful strategy found in the study, the variance was initially set equal to 0.5% of the span (upper limit to lower limit) of each parameter which was incremented by a factor

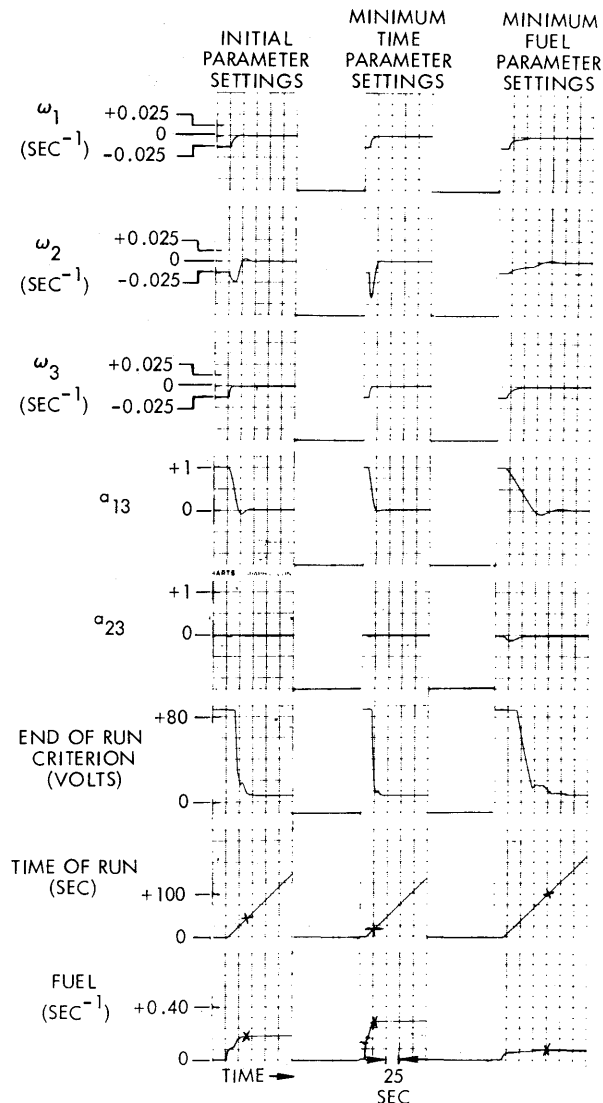


Figure 6. Comparison of minimum time and minimum fuel optimal solution with pre-optimization solution.

of 1.02 every trial until the variance reached 50% of the span. This strategy assumes that the probability of a further improvement is highest near the local optimum and decreases linearly as the distance from the local optimum. In nearly every search for which the global optimization strategy was used, improvements were found.

4. With 9 free parameters 300 to 500 runs were made, with a limit of 100 or 200 runs in the global search subroutine. With 2 solutions per second, the total time was approximately 4 to 6 minutes per optimization. It should be noted that with a modern high speed iterative analog computer, this figure could be reduced by one to two orders of magnitude.

A NOTE ON CONVERGENCE

It has been stated by Korn⁷ that the random search technique will converge whenever the gradient technique does. Clearly, for the local optimization algorithm, convergence can be assured since the strategy results in a sequence of criterion function values which is monotonically decreasing and bounded from below by zero. However, the rate of convergence is another matter. Rastrigin, who published one of the early papers on random search optimization,¹¹ has also investigated its convergence properties.^{12, 13} In the case of unimodal criterion functions, where constant F contours are hyperspheres in the parameter space, he shows that the mean rate of progress of the random search method in the gradient direction exceeds that of the steepest descent method when more than 3 parameters are involved. However, no such proof is available for the nonlinear case.

Global search, which in the limit samples the parameter space everywhere, will converge to the global optimum with probability one. However, any computer implementation is finite and therefore cannot insure the location of the global optimum.

CONCLUSIONS

Parameter optimization by sequential random perturbation is an efficient and easily programmed technique for the optimization of nonlinear dynamic systems. The technique is well suited to hybrid computation.

REFERENCES

1. S. H. Brooks, "A Comparison of Maximum Seeking Methods," *Operations Research*, vol. 7, pp. 430-57 (July 1959).
2. D. J. Wilde, *Optimum Seeking Methods*, Prentice-Hall, Englewood Cliffs, N.J., 1964.
3. S. H. Brooks, "A Discussion of Random Methods for Seeking Maxima," *Operations Research*, vol. 6, pp. 244-51 (March 1958).
4. J. K. Munson and A. I. Rubin, "Optimization by Random Search on the Analog Computer," *IRE Trans. on Elec. Computers*, vol. EC-8, pp. 200-203 (June 1959).
5. R. R. Favreau and R. G. Franks, "Statistical Optimization," *Proc. 2nd Intern. Analog Computer Conference*, 1958.
6. B. A. Mitchell, "A Hybrid Analog-Digital Parameter Optimizer for Astrac-II," *Proc. AFIPS*

Spring Joint Computer Conference, vol. 25, pp. 271–85 (1964).

7. G. A. Korn, *Random Process Simulation and Measurements*, McGraw-Hill, New York, 1966, pp. 207–10.

8. A. E. Sabroff, et al, "Investigation of the Acquisition Problem in Satellite Attitude Control," Air Force Technical Report AF FDL-TR-65-115 (Dec. 1965).

9. R. E. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, Princeton, N.J., 1961.

10. R. J. Adams and A. Y. Lew, "Modified

Sequential Random Search Using a Hybrid Computer," University of Southern California, Electrical Engineering Department report (May 1966).

11. L. A. Rastrigin, "External Control by the Method of Random Scanning," *Automation and Remote Control*, vol. 21, pp. 891–96 (1960).

12. —, "The Convergence of the Random Search Method in the External Control of a Many-Parameter System," *ibid*, vol. 24, pp. 1337–42 (1963).

13. L. S. Gurin and L. A. Rastrigin, "Convergence of the Random Search Method in the Presence of Noise," *ibid*, vol. 26, pp. 1505–11 (1965).

A PARAMETRIC GRAPHICAL DISPLAY TECHNIQUE FOR ON-LINE USE

M. L. Dertouzos and H. L. Graham

*Massachusetts Institute of Technology
Cambridge, Massachusetts*

INTRODUCTION

Graphical displays are gaining unquestioned importance¹ in the growing field of communication between man and digital computers. The development of time-shared digital computers² further accentuates the need for widely-used graphical interaction within the framework of certain economic constraints. Typically, these constraints involve the use of relatively low telephone-channel data rates, and the temporary storage of graphical data at the display site. User requirements dictate both an input and output ability of alphanumeric and graphical data* for typical applications. A typical time-shared application, which gave rise to the technique and prototype described in this paper, involves the online design of electronic circuits.^{3, 4} Here, a designer communicates graphically a circuit to the computer, visually observes circuit-analysis results in the form of curves, and modifies on-line circuit parameters and topology in order to improve circuit performance.

The display technique to be presented involves only the output of graphical data. It is normally combined with a device such as a teletypewriter or

* This distinction between alphanumeric and graphical data is one of efficiency, since character generation and graphical-segment generation could be accomplished by the same mechanism:

a light-pen for graphical data input. Main objectives in developing this approach have been the display of complex curves in terms of relatively few computer commands, and the evolution of a display system sufficiently simple and inexpensive to accompany a type of a time-shared installation.

First, the principle of operation is described. Discussion of certain system-design decisions is followed by the way in which the experimental prototype was implemented. Experimental results, relevant software, and conclusions complete the paper.

PRINCIPLE OF OPERATION

A block diagram of the basic elements used in this technique is shown in Fig. 1. Visual data is provided by an x - y graphical display device such as a storage-type cathode ray tube driven by waveforms $x(t)$ and $y(t)$, where t is time. These waveforms are the outputs of two linear networks, T_x and T_y , char-

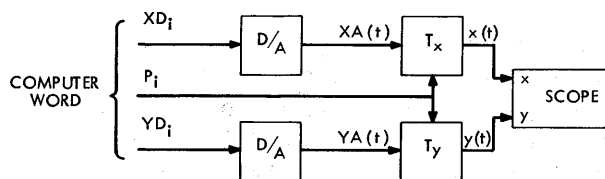


Figure 1. Basic configuration.

acterized by unit-step responses $T_x(t)$ and $T_y(t)$, respectively, which are constrained to have unity steady-state gain, that is $T_x(\infty) = 1$ and $T_y(\infty) = 1$. Signals $XA(t)$ and $YA(t)$ are the outputs of digital-to-analog converters which convert two parts, XD_i and YD_i , of the computer output from digital into analog form. A third part, P_i , of the computer output controls the *parameters* of networks T_x and T_y .

Assume that at time, t_k , the linear networks have attained steady state. As a consequence, outputs of T_x and T_y will be $x(t_k) = XA(t_k)$ and $y(t_k) = YA(t_k)$, thus establishing at time, t_k , a point $[XA(t_k), YA(t_k)]$ on the CRT. Suppose that at $t = t_k$ the computer output changes, so that

$$XA(t) = XA(t_k) + \Delta X u_{-1}(t - t_k) \quad (1)$$

$$YA(t) = YA(t_k) + \Delta Y u_{-1}(t - t_k) \quad (2)$$

for $t > t_k$, where $u_{-1}(t)$ is the unit step. Signals $x(t)$ and $y(t)$ will then be, by the linearity of T_x and T_y , for $t \geq t_k$

$$x(t) = XA(t_k) + \Delta X T_x(t - t_k) \quad (3)$$

$$y(t) = YA(t_k) + \Delta Y T_y(t - t_k) \quad (4)$$

Since T_x and T_y are constrained to have unity steady-stage gain, the final values of $x(t)$ and $y(t)$ at some time t_{k+1} , where $t_{k+1} - t_k$ is much greater than the time constants of T_x and T_y , will be

$$x(t_{k+1}) = XA(t_k) + \Delta X \quad (5)$$

$$y(t_{k+1}) = YA(t_k) + \Delta Y \quad (6)$$

thus, establishing a point $[XA(t_k) + \Delta X, YA(t_k) + \Delta Y]$ at time t_{k+1} on the CRT.

What is of interest here is the trajectory $f(x, y) = 0$ —resulting from elimination of t in Eqs. (3) and (4)—between these two points. This trajectory will depend upon the nature of networks T_x and T_y and on the way in which their internal parameters are controlled by P_i . For instance, in the special case where T_x is identical to T_y , Eqs. (3) and (4) give the following trajectory:

$$y = Y(t_k) + \frac{\Delta Y}{\Delta X} (x - X(t_k)) \quad (7)$$

which is, as expected, a straight line. In general, when T_x and T_y are not identical, the trajectory will be some type of curve segment dependent upon the class and parameters of T_x and T_y .

It is, therefore, desirable that T_x and T_y be properly chosen, so that a large and useful class of curve segments will be available through appropriate con-

trol of network parameters with P_i . A complex curve may then be synthesized as a composition of these basic curve segments.

SELECTION OF NETWORKS T_x AND T_y

Let T_x and T_y have step responses which are rising exponentials, i.e.,

$$T_x(t) = (1 - e^{-\sigma_x t}) u_{-1}(t) \quad (8)$$

$$T_y(t) = (1 - e^{-\sigma_y t}) u_{-1}(t) \quad (9)$$

where σ_x, σ_y are positive and $u_{-1}(t)$ is the unit step. If $XA(t_k) = X_0$, $YA(t_k) = Y_0$, $\Delta X = (X_1 - X_0)$, and $\Delta Y = (Y_1 - Y_0)$, then by Eqs. (3) and (4), signals $x(t)$ and $y(t)$ will be

$$x(t) = (X_1 + (X_0 - X_1)e^{-\sigma_x(t-t_k)}) u_{-1}(t - t_k) \quad (10)$$

$$y(t) = (Y_1 + (Y_0 - Y_1)e^{-\sigma_y(t-t_k)}) u_{-1}(t - t_k) \quad (11)$$

for $t \geq t_k$. Eliminating time between Eqs. (10) and (11) gives, between points (X_0, Y_0) and (X_1, Y_1) , the following trajectory:

$$y = Y_1 + (Y_0 - Y_1) \left[\frac{(x - X_1)}{(X_0 - X_1)} \right]^{\frac{\sigma_y}{\sigma_x}} \quad (12)$$

By varying the ratio of natural frequencies, σ_y/σ_x , the class of curve segments connecting points (X_0, Y_0) and (X_1, Y_1) is given by Eq. (12). Various members of this class are shown in Fig. 2, for $X_0 = Y_0 = 0$ and $X_1 = Y_1 = 1$. As shown, it is possible to obtain a large, well-spaced family of curves by properly varying σ_y/σ_x . This set of curve segments, however, has a major disadvantage. The final slope of a segment is easily shown to be:

$$\left. \frac{dy}{dx} \right|_{x=X_1} \begin{cases} 0 & \text{for } \sigma_y > \sigma_x \\ (Y_0 - Y_1)/(X_0 - X_1) & \text{for } \sigma_y = \sigma_x \\ \infty & \text{for } \sigma_y < \sigma_x \end{cases} \quad (13)$$

That is, the final slope is either zero or infinity except for the special case when the segment is a straight line. This constraint imposes the severe limitation of slope discontinuities between adjacent curve segments.

It appears, then, that besides the determination of steady-state coordinates, the selected networks should have at least two degrees of freedom in order

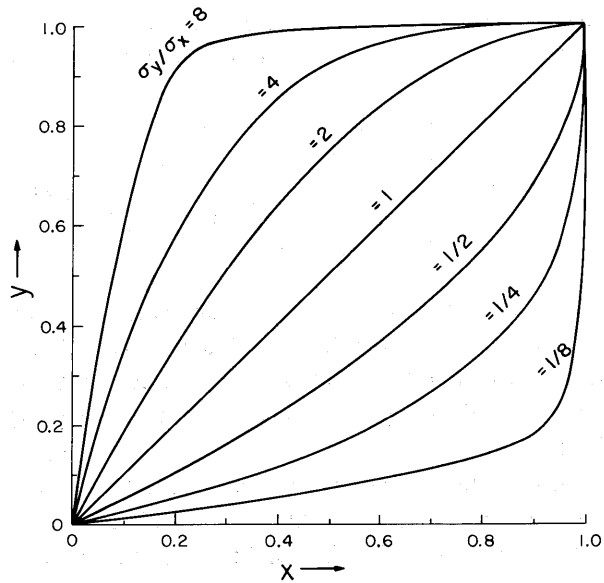


Figure 2. Family of curves obtained from simple exponential realization.

to specify either initial or final slope, and in order to yield, for each such slope, a class of varying-curvature segments. On this basis, two alternative realizations of T_x and T_y are investigated next.

Polynomial Realization

Let $T_x(t)$ and $T_y(t)$ be of the form

$$T_x(t) = (1 - e^{-\sigma t})u_{-1}(t) \tag{14}$$

$$T_y(t) = (\alpha + \beta e^{-\sigma t} + \gamma e^{-2\sigma t} + \delta e^{-3\sigma t})u_{-1}(t) \tag{15}$$

where $\alpha + \beta + \gamma + \delta$ is constrained to be the initial value of y , in this case 0, and α is constrained to be the final value, in this case 1. The initial slope of the resulting trajectory from (X_0, Y_0) to (X_1, Y_1) is

$$\left. \frac{dy}{dx} \right|_{x=X_0} = \frac{\beta + 2\gamma + 3\delta}{X_0 - X_1} \tag{16}$$

and the resulting trajectory curve is a third-order polynomial of the form

$$y(x) = Ax^3 + Bx^2 + Cx + D \tag{17}$$

Defining initial slope M_0 and class index $Q = \gamma + \delta$, yields for the coefficients of Eq. (17) the following:

$$A = (Q - Y_c + M_0X_c)/X_c^3 \tag{18}$$

$$B = (2Q + M_0X_c - Y_c) / (X_c^2 - 3X_1(Q - Y_c + M_0X_c) / X_c^3) \tag{19}$$

$$C = (Q + Y_c) / X_c - 2X_1(2Q + M_0X_c - Y_c) / (X_c^2 + 3X_1(Q - Y_c + M_0X_c) / X_c^3) \tag{20}$$

$$D = Y_1 - X_1(Q + Y_c) / X_c + X_1^2(2Q + M_0X_c - Y_c) / (X_c^2 - 3X_1(Q - Y_c + M_0X_c) / X_c^3) \tag{21}$$

where $X_c = X_1 - X_0$ and $Y_c = Y_1 - Y_0$. Some representative curves generated by varying M_0 and Q are shown in Figs. 3 and 4. As seen from these figures, this approach permits specification of initial slope of a given curve segment to match the final slope of the previous segment. A good "fit" to the desired curve can then be obtained by varying Q . Values of α , β , γ and δ are determined by the computer, and are converted into appropriate commands through P_i . More degrees of freedom are available by varying γ and δ independently instead of attempting to match slopes.

Exponential Approach

An alternative realization is given by

$$T_x(t) = (1 - (\alpha_x e^{-\sigma_0 t} + (1 - \alpha_x) e^{-\sigma_x t}))u_{-1}(t) \tag{22}$$

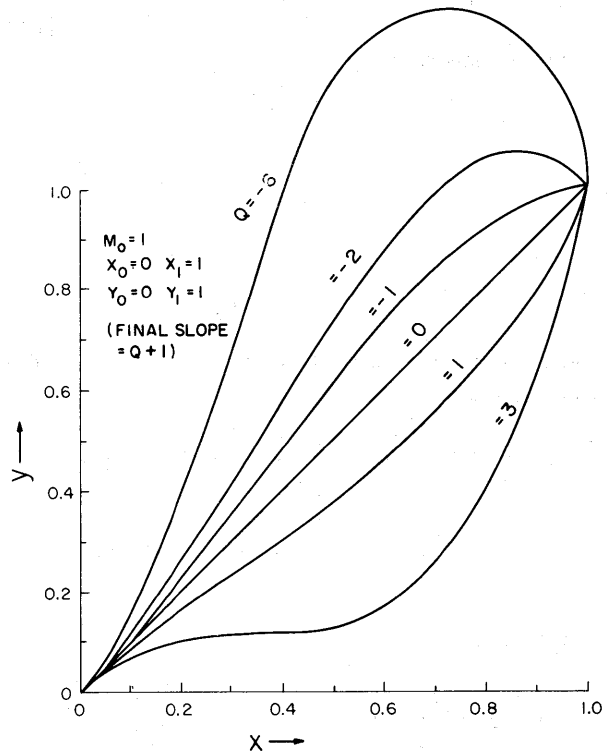


Figure 3. Curve segments from polynomial realization.

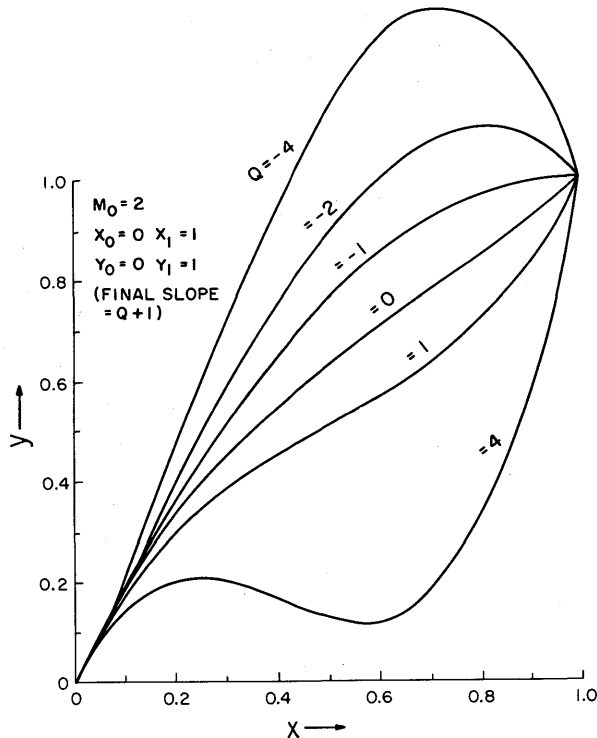


Figure 4. Curve segments from polynomial realization.

$$T_y(t) = (1 - (\alpha_x e^{-\sigma_0 t} + (1 - \alpha_y) e^{-\sigma_y t})) u_{-1}(t) \quad (23)$$

under the constraints

$$\begin{aligned} 0 &\leq \alpha_x \leq 1 \\ 0 &\leq \alpha_y \leq 1 \\ \alpha_x < 1 &= \alpha_y = 1 \\ \alpha_y < 1 &= \alpha_x = 1 \\ \sigma_x > \sigma_0 &\text{ and } \sigma_y > \sigma_0 \end{aligned}$$

The resulting trajectory from (X_0, Y_0) to (X_1, Y_1) is of the form

$$y = Y_1 + (Y_0 - Y_1) \left(\frac{\alpha_y (x - X_1)}{X_0 - X_1} + (1 - \alpha_y) \left(\frac{x - X_1}{X_0 - X_1} \right)^{\frac{\sigma_y}{\sigma_0}} \right) \text{ for } \alpha_x = 1 \text{ and } \alpha_y \leq 1 \quad (24)$$

$$x = X_1 + (X_0 - X_1)$$

$$\left(\frac{\alpha_x (y - Y_1)}{Y_0 - Y_1} + (1 - \alpha_x) \left(\frac{y - Y_1}{Y_0 - Y_1} \right)^{\frac{\sigma_x}{\sigma_0}} \right) \text{ for } \alpha_y = 1 \text{ and } \alpha_x \leq 1 \quad (25)$$

Some typical members of this class are shown in Figs. 5 and 6. The final slope m_f of any such segment is

$$m_f = \begin{cases} \alpha_y \left(\frac{Y_0 - Y_1}{X_0 - X_1} \right) & \text{for } \alpha_x = 1 \\ & \alpha_y \leq 1 \\ \frac{1}{\alpha_x} \left(\frac{Y_0 - Y_1}{X_0 - X_1} \right) & \text{for } \alpha_y = 1 \\ & \alpha_x \leq 1 \end{cases} \quad (26)$$

From Eq. (26), final slope is determined only by parameter α_x or α_y and by coordinate values. Therefore, a large family of curves with a given final slope can be obtained by fixing x and y and by varying the ratio of natural frequencies as shown in Fig. 6.

Both of the above realizations can display complex curves with smoothly connected segments. While the first realization yields a larger class of curve segments, it has two distinct disadvantages when compared to the latter: (1) computing time necessary to select an optimum set of segments to match a given curve is far greater, and (2) implementation is much more complicated and expensive. For example, the order of three times as much equipment is required, and there exist timing problems and an increased tolerance sensitivity. The prototype was, therefore, implemented with two exponential-type networks, capable of slope matching and class indexing, given by Eqs. (22) and (23). Further implementation considerations follow.

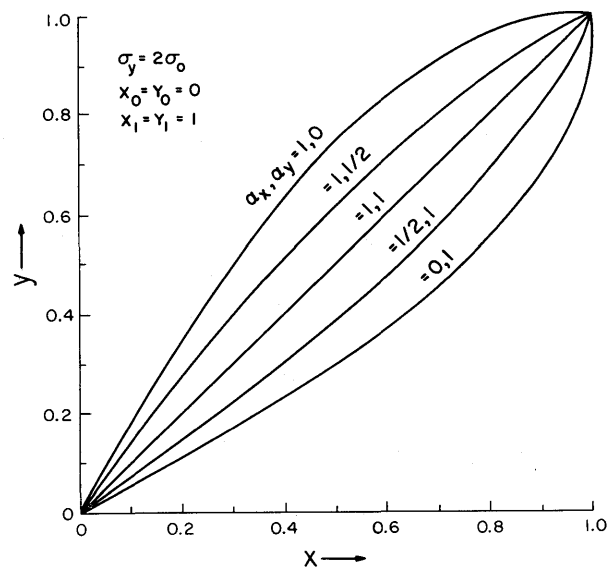


Figure 5. Curve segments obtained from exponential realization.

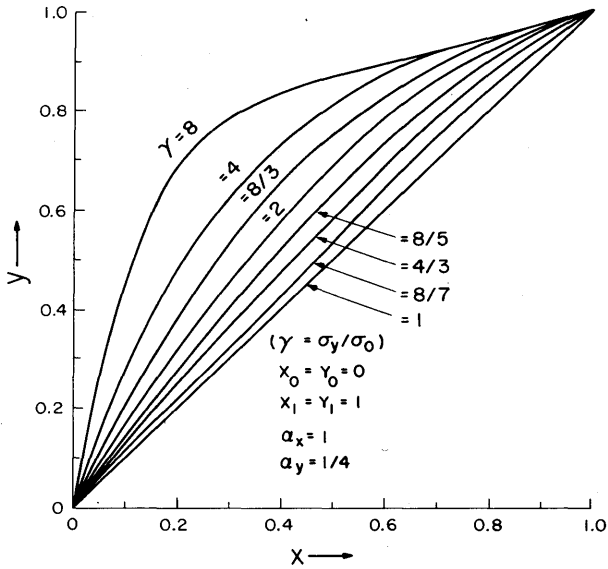


Figure 6. Curve segments with the same final slope.

IMPLEMENTATION

The experimental prototype was designed for use in conjunction with a remote teletypewriter console of the time-shared computer facilities at Project MAC. Digital information specifying a curve segment is available at the console in series-parallel

form, as four* bits per character. Bit requirements of each curve segment are as follows: six bits each for XD and YD , six bits for α_x and α_y , five bits for γ (the ratio of natural frequencies), and a single bit for designating whether the beam is to be ON or OFF. Thus, 24 bits, or 6 characters are needed to specify each curve segment.

The basic components of the system are shown in the block diagram of Fig. 7. Functions of the major components are as follows.

The *interface* has three functions: (1) It converts bit information from the console to signal levels compatible with the system logic. (2) It generates a strobe pulse coincident with the console signals for sampling purposes. (3) It recognizes the delimiter signal (which defines the completion of a segment description) and generates a delimiter pulse coincident with this signal.

The *storage registers* retain the curve parameters supplied by the computer. The *word designator* "points" to that por-

* Actually, 8 bits are sent to the teletypewriter for each character. However, all combinations of only 4 bits are available.

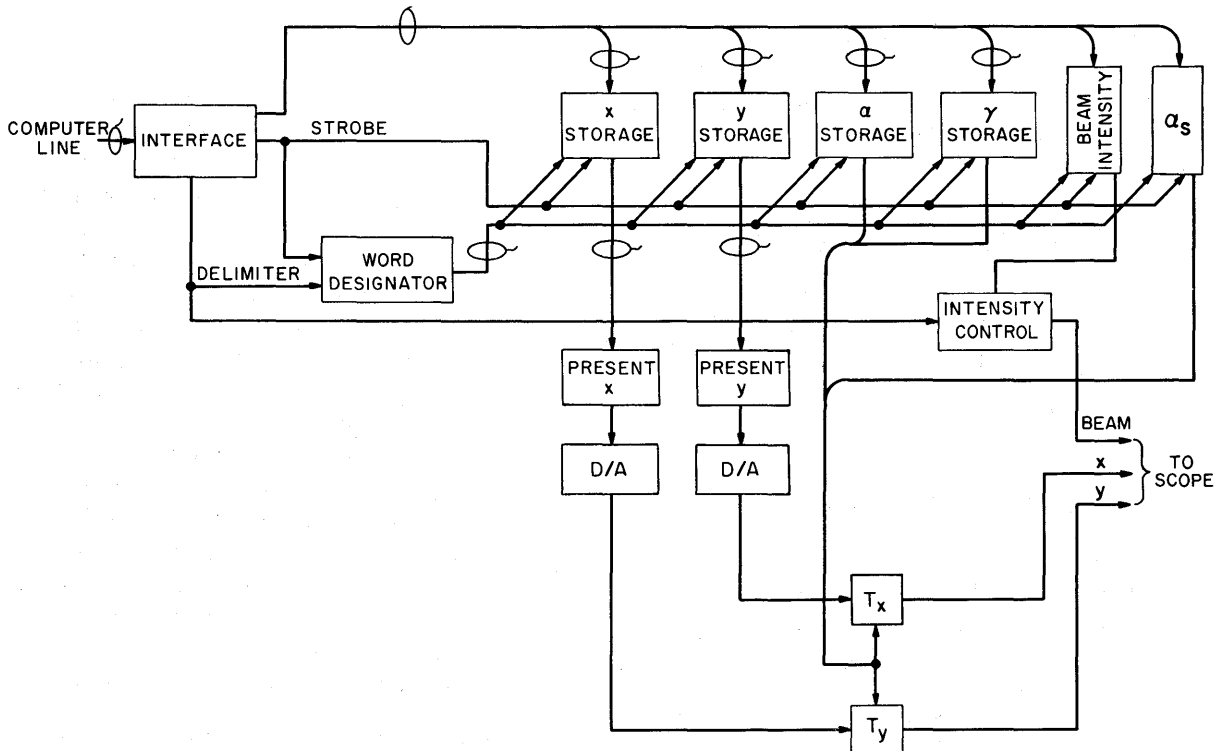


Figure 7. Block diagram of complete system.

tion of the storage register which is to be filled by each set of bits. It is advanced to sequential portions of the storage register by the strobe pulse and is reset by the delimiter pulse.

The *D/A converters* along with their associated storage, convert the coordinate value to analog form. These converters are loaded by the delimiter pulse.

If the beam-intensity bit is ON, the *intensity control* increases the intensity of the trace while the curve segment is being plotted.

A Tektronix type 564 storage oscilloscope is used as the display device.

Implementation of T_x and T_y follows the configuration decided upon in the previous section, and is shown in Fig. 8. Note that due to the constraints on α_x and α_y only one of the variable resistors of Fig. 8 is in use for plotting any given segment. Likewise, only one set of the multipliers is used for plotting any segment. Thus, the variable resistor, an α multiplier and a $(1-\alpha)$ multiplier are time-shared between T_x and T_y as shown in Fig. 9. The switch control, α_s , which is one of the bits used to specify α , differentiates between the two cases ($\alpha_x = 1, \alpha_y \leq 1$) and ($\alpha_y = 1, \alpha_x \leq 1$). The multipliers are realized with a resistive divider scheme, and have variable "gain" from 0 to 1 in steps of $1/16$. The buffers and the adder are differential amplifiers with unity feedback.

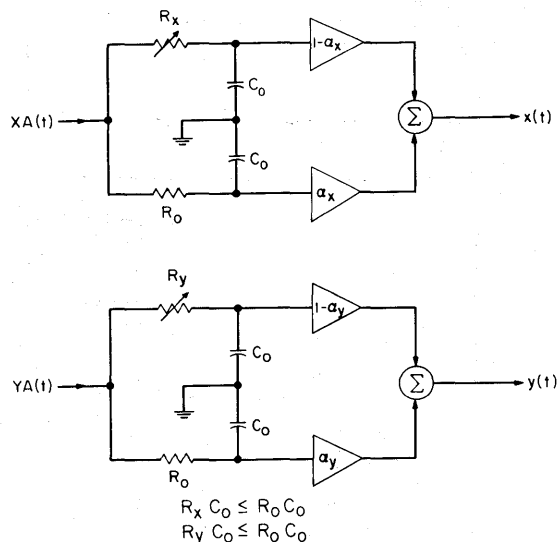


Figure 8. Configuration of T_x and T_y .

As shown in Fig. 9, the variable resistance is obtained by operating switches across resistors in a series string. For reasons of economy, magnetic reed switches were used here and whenever a floating switch was needed. These switches have a switching time of 1 msec which is adequate for present data rates. Solid-state switches may be easily substituted if the device is to be used with a higher data-rate channel.

SOFTWARE TRANSLATOR

The process of converting a desired curve from point-by-point description to a series of segments within the class, realizable by T_x and T_y , is the task of a software translator.

In the case of online design of electronic circuits³ the curve to be displayed is available within the computer as a set of closely spaced points or ordered pairs (X_i, Y_i) . The translator software converts this set of points to a set of ordered quadruples $(XD_i, YD_i, \alpha, \gamma)$, which are display commands.

A chosen performance index matches the given curve with the minimum possible number of segments within a given maximum allowable error. Wherever possible, slopes of connecting segments are matched. For simplicity, the translator algorithm has been divided into several steps.

1. Initially the curve is scanned to locate all local maximum and minimum points in both the x and y directions. Since all segments are single-valued in both x and y (except for the trivial cases of vertical and horizontal lines), no single segment can contain such a maximum or minimum. Thus the curve is matched between successive pairs of these points with a minimum number of segments. All these points are stored in an array, P .
2. Next, a search is conducted for curve segments with the realizable class that match successive points in P with minimum error. This search, which is binary, locates the smallest number of segments that will match the curve between each pair of points in P within a maximum specified error.* For each

* Absolute value of the error in the $x(y)$ direction is used as error measure for segments that are functions of $y(x)$.

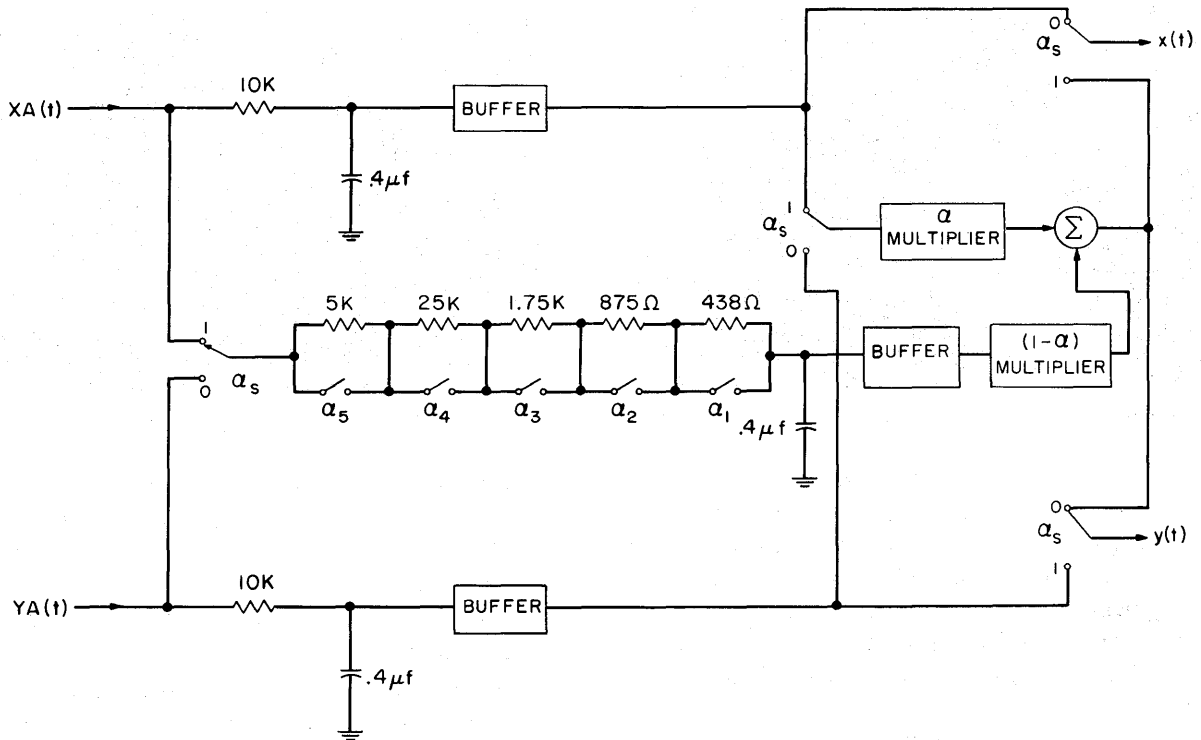


Figure 9. Schematic of reduced T_x and T_y .

segment α is set to match the slope of the previous segment or, if this is not possible, to match the slope of the given curve.

3. The computer words found in Step 2 above are then mapped into the corresponding symbols to be sent to the console.

This algorithm was implemented in the AED version of Algol as used by Project MAC.

EXPERIMENTAL RESULTS

Used in connection with CIRCAL,⁴ the language for on-line design of electronic circuits, the prototype hardware and the translator software gave rise to results shown in Figs. 10, 11, 12, and 13. Figure 10 shows output voltage versus time for a tunnel diode circuit. Four segments were used to compose this curve. No display time was spent in labeling and dimensioning curves. If necessary, this information can be supplied by the typewriter upon

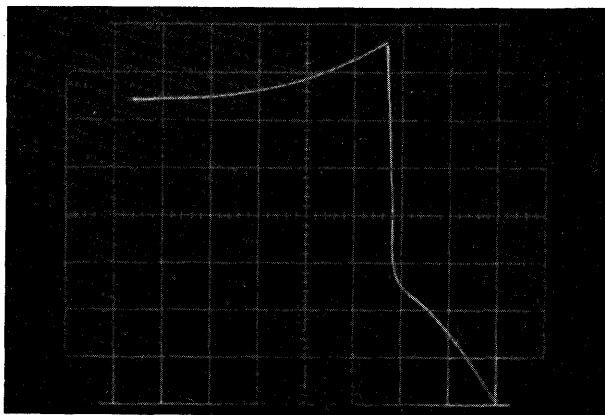


Figure 10. Voltage response of tunnel diode circuit (0.5% fit).

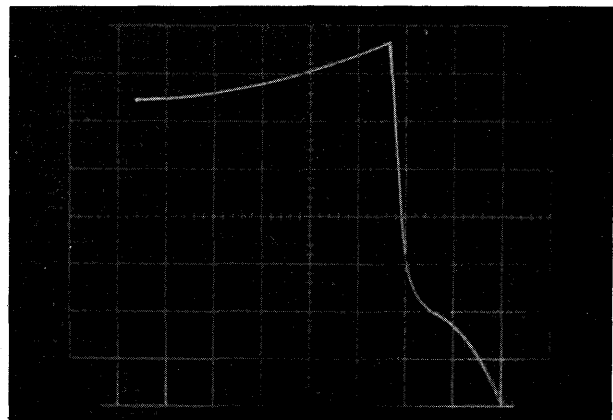


Figure 11. Voltage response of tunnel diode circuit (3% fit).

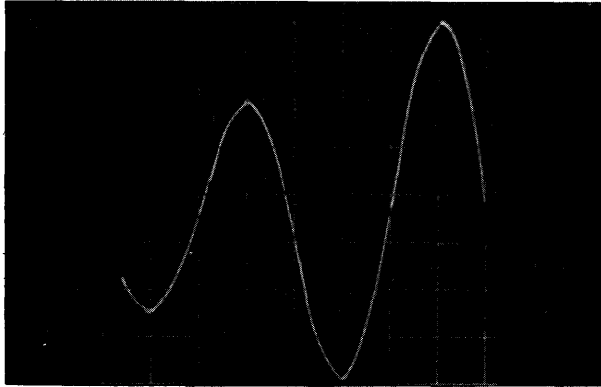


Figure 12. Step response of a damped L-C circuit.

request. The translator operated for a fit of 0.5%. "Fit" here is the ratio of error measure, defined in the preceding section, to full scale. Decreasing the desired accuracy of fit to 3% reduced the number of segments to three as shown in Fig. 11 for the same curve. Computing time spent in translation was 0.5 seconds for a 0.5% fit and 0.3 seconds for the 3% fit.

Figures 13 and 14 show other curves resulting from this display. Observe that in Fig. 13 the storage property of the scope permits the superposition of any number of curves for comparative study. Figure 14 shows an automobile silhouette drawn with 26 segments. These segments are shown in Fig. 15. This number of segments can be contrasted with possibly a few hundred points necessary for a piecewise-linear approximation or a point-by-point plot of the same drawing.

CONCLUSIONS

The main feature of this display technique is the utilization of few computer words for the display of

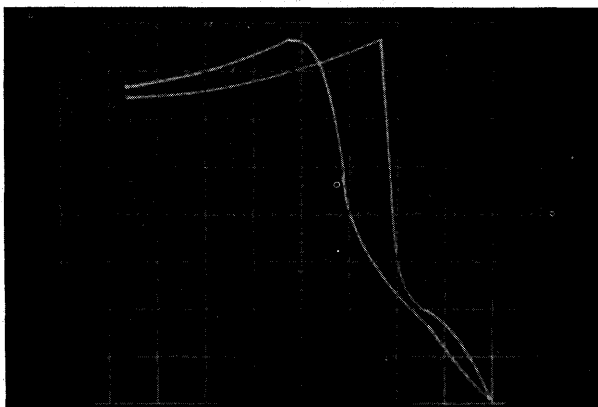


Figure 13. Superimposed voltage responses.

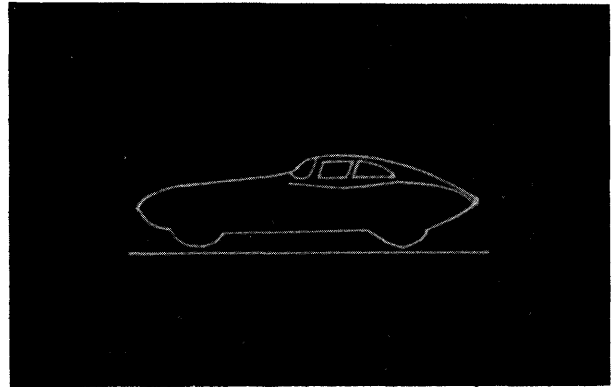


Figure 14. Automobile silhouette.

a complex curve. Advantages resulting from this feature are small storage requirements for the translated curves and a relatively fast display time. The main penalty is the computational effort necessary for translation of a given curve into appropriate display commands.

One obvious conclusion is that the display should be more advantageous in cases where standard parts are repeatedly used. In such cases, an initial translation would be successfully justified after repeated use of the translated data. Besides the above class of applications, the prototype has proven to be useful in cases of one-time curve plotting. Used in the on-line design of electronic circuits, the fraction of a second spent in curve translating can be justified both economically and temporally, since it represents but a small fraction of the computation effort allocated to circuit design. The advantage of having a visual display of the analyzed data in a typical display time interval of 2-3 seconds by far overrides the inconvenience of using the teletypewriter as a plotting tool at the expense of 3-4 minutes of plotting time and significant accuracy.

ACKNOWLEDGMENT

This research was sponsored in part by the National Aeronautics and Space Administration under contract number NsG-496 (Part). Work reported herein was supported (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 Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

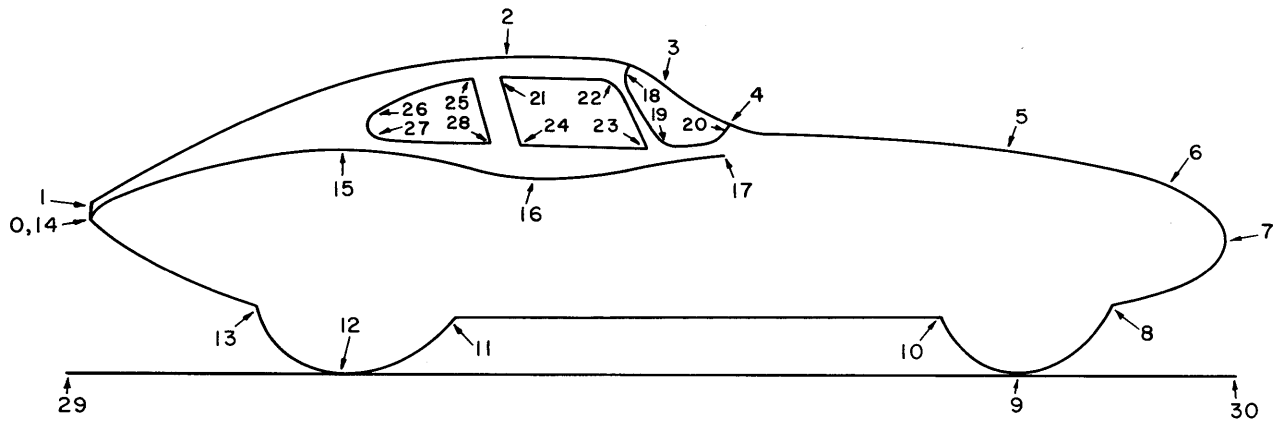


Figure 15. Command points for plot of Jaguar.

REFERENCES

1. Nilo Lindgren, "Human Factors in Engineering," *IEEE Spectrum*, vol. 3, no. 4 (Apr. 1966).
2. MIT Project MAC Progress Report, July 1964.
3. J. F. Reintjes, and M. L. Dertouzos, "Compu-

ter Aided Design of Electronic Circuits," Wincon Conference, Feb. 1966, Los Angeles, Calif.

4. M. L. Dertouzos and C. W. Therrien, "CIRCAL: On-Line Analysis of Electronic Networks," Report ESL-R-248, MIT Electronic Systems Laboratory (Oct. 1965).

A SYSTEM FOR TIME-SHARING GRAPHIC CONSOLES

James R. Kennedy

*Lockheed-Georgia Company
Marietta, Georgia*

INTRODUCTION

There is a large class of problems whose solution statements, in their most natural form, involve not only operation of verbal algorithms on easily defined data, but also provide assistance and prompting to a man "in-the-loop" that enables him to give information or data which may otherwise be inaccessible or not easily described. The advent of developments in the area of graphic hardware provided the necessary environment which allows for implementation of problem-solution statements which include provisions for human intervention. In order to economically justify human intervention, it is necessary, first, that there be sufficient problems of this type requiring a solution. Once this demand becomes apparent, it is further necessary that a system for sharing a single high-speed computer among several human operators be made available. Such a system, which provides for time-sharing (with good response characteristics) a single central processing unit among several graphic display consoles, is the subject of this paper.

GRAPHIC INTERACTION

From an intuitive point of view, graphic interaction is fairly well understood. It consists of a close interplay between a human operator at a graphic console and programs written for the pur-

pose of carrying out the computer's role in the interaction. The operator at the console has, as his means of communications, several methods for generating hardware interrupts which can be examined by the programs for syntactic and semantic information on which to base a reaction. Some of the commonly known ways of generating these interrupts are by pressing one of several buttons on a keyboard panel, pressing a button on a light-pen, stepping on a foot pedal, or holding a light-pen over some portion of the display cathode-ray tube at a time when its electron beam is striking the face of the tube in proximity to the position of the light-pen. This last method of generating an interrupt can be referred to as a light-pen strike or, more simply, a "pen strike."

By these and other methods, therefore, the operator generates a stream of interrupts, the meaning of which must be interpreted by programs. For example, programs are written such that when a particular button is pressed, an associated subroutine is called to perform the function implied by the button. Another example is found in the "light button." The light button concept is brought into play by the operator in the following way: The operator depresses the light-pen button causing an interrupt which signals that he wishes to point to some light on the CRT with the light-pen; then the operator moves the pen near a marker of some sort being displayed on the CRT face. This action causes

a pen strike to be generated and interpreted in a way analogous to an interrupt caused by a button being depressed; that is, the program analyzes the position of the light which caused the pen strike or obtains sufficient information in some other way to arrive at a correspondence with a particular sub-routine which is then executed.

CPU interrupts are classified as follows: They are either caused as a result of some action on the part of the operator or generated as a result of program action. For example, in order for a system to understand the full meaning of a single depression of the light-pen button and a single pointing action on the part of the operator, it may generate a sequence of possibly four or five interrupts to be interrogated—each supplying a small portion of the total meaning. During the interaction, both system and application programs are brought into play in a random fashion depending on the whims of the console operator. Since system reaction to interrupts is very rapid compared to the reaction speed of a man at the console, one finds that the computer is usually idle and waiting for the next set of instructions in the form of interrupts.

A PROTOTYPE EXPERIMENT

In January of 1965, the Lockheed-Georgia Research Laboratory obtained a UNIVAC 418 Computer and a DEC 340 Precision Incremental Display for the purpose of studying man-computer graphics¹ systems. A "Sketchpad"²-like drawing program had already been outlined and the coding virtually completed. In the following months, this program and others formed the basis for a series of subjective human factors studies and was a training ground for both systems programmers and specialists in various fields of aircraft design. After many modifications and improvements on the original design, the basic program, which was three-dimensional³ from the beginning, was stripped apart to form a two-dimensional drawing system. This system was then expanded in its construction capabilities and refined internally to give a very efficient system for forming a generalized two-dimensional list data structure (containing geometric properties) and controlling subprogram operators which generated display information based on this data structure and made insertions, deletions, and modifications such as translation and expansion.

Coincident with the emergence of the two-dimensional drawing capability was the develop-

ment of an application program which could perform calculations based on the two-dimensional data structure for the purpose of generating commands to drive a numerically controlled milling machine. This application program, whose output resulted in a milled part of somewhat arbitrary shape, therefore formed the mechanism by which the Lockheed-Georgia Company was able to make an intimate study of the relationship between a human operator and a highly interactive graphic control system.

Among the more important results of this study were the following:

1. In general, production application programs would be too large for permanent core residency.
2. Application and graphic system routines had to be structured such that CPU control did not hang up within a subprogram waiting for further external information to continue its task.
3. Display beam-driving commands are inefficient when their storage is structured in the form of sequential core-resident arrays.
4. The concept of program and data "state" must be implemented and organized with care.
5. Of the two sets of information that are tending to grow in size as the interaction continues (listed data structure and display beam driving commands), the display commands tend to occupy the largest amount of storage.

Based on these findings and justified by adequate requirements for a large-scale production system similar to the prototype, extensive equipment evaluations were performed and specifications for a system to time-share multiple consoles were drawn up.

SYSTEM SPECIFICATIONS

Hardware

In the fall of 1965, a firm order was placed for a CDC 3300 computer configured to include a Digraphics display system with three 22" CRT devices⁴ for graphic I/O. (See Fig. 1 for CDC hardware configuration.) Driven by a controller, relatively static picture information displayed on the three CRT's is refreshed in an off-line fashion from a six-track drum. More dynamic information is

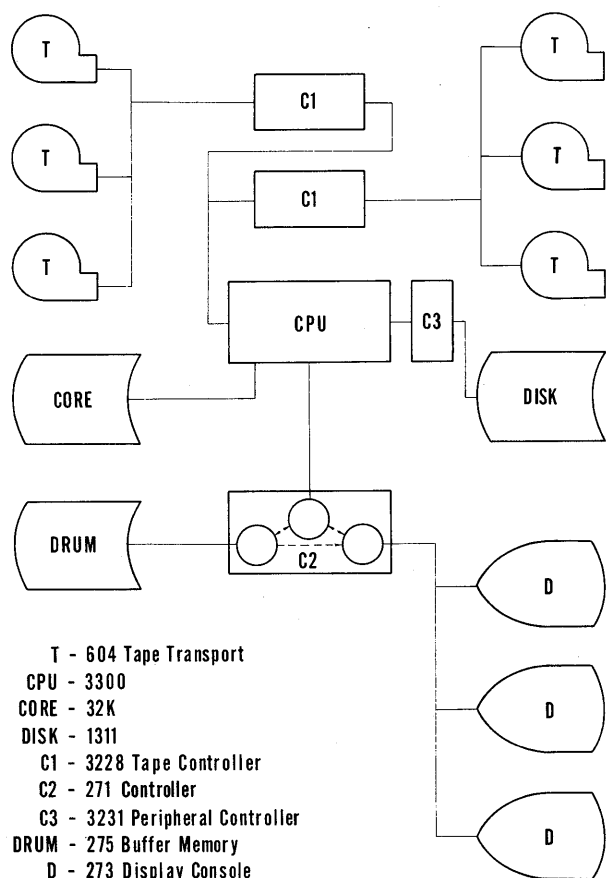


Figure 1. CDC Hardware configuration.

routed directly from core to each of the displays. Since one drum revolution requires 33 milliseconds, and because each of the six tracks can contain 10,000 words of display information, large amounts of information can be displayed "flicker free" on each CRT. Controller commands can be stored on the drum intermixed with CRT beam-driving commands in order to switch display control from one track to another. If switching is not required, each CRT could be driven from a single track thus leaving three tracks (or 30K) of high speed peripheral storage for other uses.

Software

Specifications for software listed the following requirements:

1. The entire system had to be interrupt oriented.
2. A sophisticated scheme for dynamic allocation of storage was a necessity.

3. Most system subprograms and all application subprograms would be dynamically relocatable and loaded into main memory only on demand. That is, nonresident subprograms reside permanently on disc or drum until a transfer to them for execution is attempted.
4. The system must be capable of monitoring the execution of all application subprograms.
5. It must recognize the possibility that an application subprogram may be frequently called by allowing the subprogram to remain resident for a temporary period after return from its initial execution. That is, decisions concerning residency must be based on dynamically changing requirements.
6. System subprograms dealing with I/O to graphic consoles must free application programs of the burden of details.
7. System interfacing with application subprograms must not interfere with the programmer's choice of coding language.
8. The system must respond to unrelated applications with equal assistance.
9. The system must be open-ended and thereby allow for future modification and extension.

SYSTEM IMPLEMENTATION

Development of the time-sharing system was brought about by a merger of already existing subprograms provided by Control Data Corporation's Digigraphics Laboratory and locally coded routines to provide disc I/O capability and a resident load function. A careful study of CDC's Function Control Program indicated that interrupt processors and byte stream generators could be extracted easily with only slight modifications to remove references to their overlay programs. This incorporation of routines which dealt intimately with the hardware meant that local systems programmers could turn their attention to the problem of implementation of system and application programs within a disc and drum environment. System routines are categorized as either resident or nonresident routines.

Resident System

Main Executive Program. Under idle conditions, main frame control resides in a program which is stack driven. MAIN loops on two tests. One test is to determine whether any function buttons (key-board buttons or light buttons) have been activated by a console operator and stacked by the appropriate interrupt processor. The other is to determine if a return has been stacked on completion of an I/O request. These tests involve two stacks which we can refer to as the button and I/O stacks.

Each n -component element⁵ on the button stack gives a numeric code which corresponds to a subroutine to be executed and the console from which the request for action came. The code number is used as an index to an array of pointers for the particular console to a table entry for the appropriate subroutine. The table entry for a routine gives its BCD name, a bit to indicate if the routine is a permanent core RESident and a pointer, PNTR. If the routine is a permanent core resident, RES is a 1. In this case, PNTR is the absolute entry point to the routine. If RES is a 0, PNTR may be nil or a pointer to a "status table" for the routine. If PNTR is nil, the routine is nonresident and must be loaded. If PNTR is nonzero, the status table must be referenced to determine whether the routine is resident or not. When RES is 0, there is an extra entry in the table which indicates WHERE the routine is to be found. Since a pointer to this table is the argument given to XEQT (explained below) to cause the routine to be executed, this table is called XEQTAB. See Fig. 2 for a typical entry.

MAIN uses the console number as an index to an array giving the beginning of Common for each console. Common for the appropriate console is set into a variable known as ACTCOM and a call to the subroutine is effected via an "execution interface" routine giving, as a single argument, a pointer to the table entry for this subroutine. MAIN also interrogates a millisecond clock for accounting purposes.

The second (I/O) stack contains elements giving absolute return addresses and the contents of machine registers, and is the path by which control returns to a routine that has requested an input/output function to be performed on the disc or drum unit.

XEQT. XEQT is a transparent interface between subroutine calls and the actual execution of the

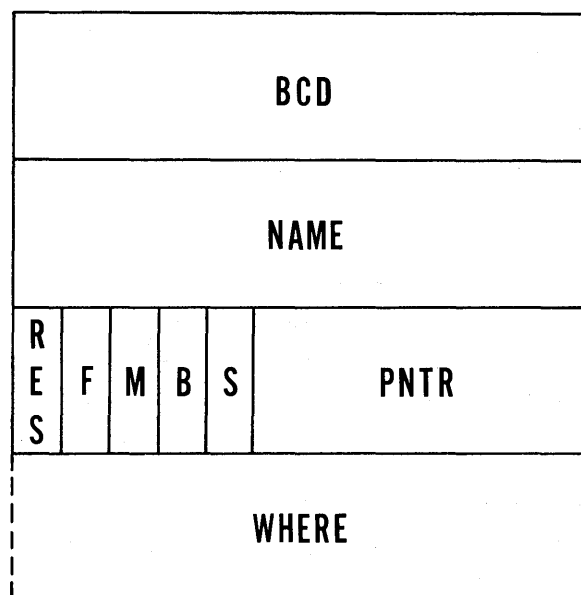


Figure 2. XEQTAB.

subroutines. The basic philosophy underlying the need for such an interface routine is that highly interactive programs will in general consist of a large set of relatively small subroutines that perform functions or subfunctions indicated by the system user. Normally, the entire program cannot be contained in core. However, with a dynamic load-at-call-time capability, those parts of the total which are required to respond to user requests can be shuttled in and out of main memory on a "demand" basis, thereby requiring only a relatively small part of main memory for program allocation at any given time.

The interface can be signaled to provide as many as 15 arguments which may be used by the subroutine in the form of dynamically allocated dimensioned arrays. The storage for each of these arrays is obtained by the interface at call-time and returned after execution of the subroutine. The actual size of the dimensioned arrays given as arguments is arbitrary and specified only in the call through the interface to the subroutine to be executed. The interface automatically provides for loading the called subroutine from either the drum or disc peripheral storage devices (using WHERE) or for entry to a routine which is already resident in core. Since both the subroutine and dynamically dimensioned arrays are returned to free storage on exit, it is necessary that any state variables or results of computations that are required by other routines after execution be placed in Common.

The compilation or assembly of a subroutine calling another routine via XEQT causes a string through the external symbol usages to be placed as the first argument (which is the name of the routine being called) in the call to XEQT. This results in the necessity for XEQT to reference directly the first argument to obtain a pointer to the table entry for the appropriate subroutine. However, there are cases where a reference to a table entry is desirable in an indirect fashion.

Suppose, for example, that a pointer to the subroutine table was stored as a component in an element. In this case, it would be necessary to fetch the component from the element and store its value into a variable declared in the fetching program. Since it is not possible to determine during compilation of the fetching routine that the component was a "procedure" component, the external symbol string does not appear. In order that XEQT may be able to reference the table for this procedure component, a special entry has been provided for this purpose.

Another entry, known as a "terminal" entry, has been provided for the purpose of concatenating subroutine calls without retaining the calling routine in core. An example of the use of this entry point would be made in the case of a control routine. The purpose of the control routine might be to examine parameters to determine which of many possible subroutines should be called. If the control routine calls the appropriate subroutine as its last logical function, it is not necessary or desirable that it remain resident. Therefore, it could make a terminal call to the appropriate subroutine, which, upon completion of execution, would return control directly to the routine which called the control routine. The control routine would have been unloaded at the time of the terminal call. Entries to XEQT which can be typed integer have also been provided. (See Fig. 3 for flow of program control when XEQT is used.)

Load Function. When a routine is found by XEQT to be nonresident, a call is made to GETPROG giving it a status table pointer, the address of console Common and the address of the call to XEQT. On return from GETPROG, XEQT is supplied with a status table pointer and caller address which correspond to a program that has been loaded.

GETPROG performs the job of providing—via FREE—an input buffer, obtaining the WHERE word and building a stack block which contains a

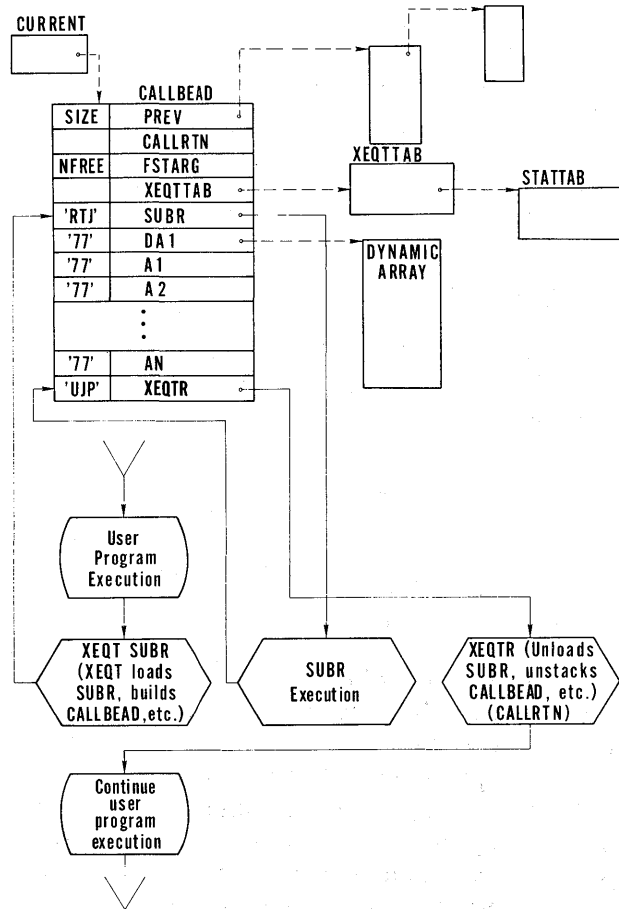


Figure 3. Program flow.

call to an input routine and a call to a loader initializing routine. A jump into the block at the input routine call is then executed carrying parameters in both the block and machine registers to the loader initializer.

The input routine stacks the input request and then tests to see if I/O is busy. If so, a jump to the idle loop in MAIN takes place where other consoles may have requests to be performed in the form of stacked buttons or return addresses resulting from I/O requests. If I/O is not busy, it is initialized and then a jump to the idle loop is executed. In either case, it is the jump to the idle loop by the input/output stacking routine which provides for processing of other requests when an I/O request forces a "wait" condition on a particular console.

After the input has been performed and control passes to the loader initializer via a "return-jump" instruction in the stacked block (which gives a pointer to a fixed word in the block), the initializer

P R I	SIZE	ENTRY
		OTHERS
		XEQTTAB
IF PRI-1		BLOCKA
IF PRI-0		MONIT
IF PRI-1		BLOCKS
IF PRI-0		NONE
IF PRI-1		MONIT
IF PRI-0		NONE

Figure 4. STATTAB.

examines the first card of the program to determine program size. This is used to get a block into which the program will be loaded. Program block address and size are put into STATTAB (Fig. 4).

The return-jump to the initializer is changed to a return-jump to a routine which will call the loader to process each card beyond the first. This routine gives the loader a relocation factor, next available Common location and a pointer to the first word of a card to be processed. When the program overlaps onto more than one drum sector or disc track, it jumps back into the stacked block at the call to the input routine. After the program has been loaded a return to XEQT is made as mentioned above.

Building and Using Tables. For each entry point defined by the load process, the loader calls an entry to LOADAID giving it the BCD name of the entry and its absolute address. If the BCD name of the entry being processed does not match the BCD name of the routine that was called, the entry is a secondary entry and a pointer to its XEQTTAB is found by searching all XEQTTAB's in the "neighborhood" of the table for the routine which was called. When it is found, a STATTAB is formed and PNTR is set. The STATTAB of the "called" entry is formed by the loader initializer before it stores program block parameters. The status table for secondary entries to a routine does not contain the program block address and size since it is unnecessary. Bit 23 is set zero for all secondary entries to indicate this

difference. In all other ways, however, the STATTAB's for both primary and secondary entries are identical.

The absolute address which corresponds to the BCD name given to LOADAID by the loader is stored in STATTAB along with a pointer to the appropriate XEQTTAB. One other entry is made in STATTAB at this time. It is a pointer to the status table for the primary entry. Each time the loader calls LOADAID giving a new entry and a BCD name, the STATTAB of the previous entry is made to point to the new entry and the new entry is made to point to the primary entry. In this way, a "ring" of entry points to the routine being loaded is kept to allow for unloading all entries at the same time and returning all STATTAB's relating to the routine.

Four other entries appear in word 2 of the XEQTTAB of each entry. Each relocatable program on peripheral storage has a set of these to be interpreted as follows: a bit to indicate frequency; a bit to indicate monitor status; a bit to indicate a breakpoint; and a four bit integer to indicate the number of dynamically allocated arrays to be supplied by XEQT. When the program is output to a peripheral device at sign-on time, the four-bit integer is picked up from a control card and inserted into the XEQTTAB as it is being formed for each entry point. The remaining bits are Booleans that are set dynamically.

When XEQT is setting up the call sequence block for the subroutine, if the integer giving the number of dynamic arrays is nonzero, it is assumed that it gives the number of arguments, beginning with the first, which supply sizes of blocks required by the routine. These block sizes are used by XEQT to acquire, from free storage, pointers to blocks of the appropriate size. These pointers are then placed in the call sequence to the routine in the position of the argument corresponding to the size of the block. In this way, the blocks can be referenced as dimensioned arrays by the routine during execution. After execution is completed, XEQT refers back to the execute table to see if any dynamic arrays were supplied. If so, the call sequences are referenced again to return the arrays to free storage.

If the bit indicating monitor status is a one, extra space can be provided in each STATTAB for the routine. This space is used to store debugging or accounting information pertinent to each entry point just prior to and immediately after execution. Two system routines, known as PREXEQM and POST-

XEQM, are called to provide pre-execution and post-execution monitoring functions. At present, these routines provide a trace of calls to and returns from an entry and give application programmers a debugging aid in the form of the number of times an entry is called and the absolute location of the program block.

When the bit indicating a break-point is set, an absolute address corresponding to a given relative address within the subroutine is referenced after loading to fetch the instruction stored there. This instruction is saved in a block on the Break Point Instruction (BPI) stack which is a string that begins in STATTAB. It is then replaced by a return jump to an element in the block. The block contains a pointer component to the appropriate status table and a return jump to a system entry called Break-Point Enter (BPE). If the instruction at the break-point is reached, BPE returns the original instruction to its proper location and builds another stack block which contains the instruction address, status table pointer and machine conditions. This block is set into a Break-Point Hold (BPH) stack and can be used as input, along with the resident program, to many debugging operators. Break-Point Return (BPR) is a system routine that maps the BPH stack block into an I/O return stack block to effect resumption of the halted routine.

The final insertion into XEQTTAB is a bit which, when set to one, indicates that the frequency of execution is known or expected to be high. After execution, if this bit is set, XEQT will not "unload" the routine from core. All insertions and tables are allowed to remain for future use. It should be noted that if the monitor status bit is a one, XEQT will allow the STATTAB entries to remain resident even though the frequency bit is a zero and the routine is unloaded. If this occurs, XEQT clears the absolute address entry pointer in STATTAB to indicate that the routine is nonresident but being monitored. Future calls to the same routine result in LOADAID using the same STATTAB's that were used on previous calls.

Considerable debugging aid is obtained by pre- and post-execution monitoring routines which print out on the console typewriter the fact that a routine is going into execution or is returning and giving its absolute core location. A manual interrupt routine has been provided to allow for console typewriter entry of the names of those routines whose execution

is to be monitored and for entry of break-point information.

For each external symbol, the loader calls LOAD-AID which searches all tables for the console and provides a pointer. If the symbol represents a permanent resident routine, the pointer is its absolute entry; if it represents a nonresident routine, the pointer gives the associated XEQTTAB.

Free Storage. Two entry points are provided for call by all system and application routines to obtain blocks of contiguous storage and to return these blocks when they are no longer needed. FREE has access to a string of all blocks of contiguous storage available for use by any program. This string is formed by the RELease N WorDs (RELNWD) entry and consists of all blocks which have been returned to free storage after use. A request for a certain sized block is a call to FREE to search the string for the smallest block which will satisfy the request. FREE then returns the unneeded portion of the block to the RELNWD entry and gives, on return, the requested portion of the block to the user program in the form of a pointer to the first word in the block.

Because even a simple action on the part of a console operator may result in many calls to free storage for different sized blocks for the purpose of loading programs and building data both permanent and temporary, storage tends to be broken into many small blocks as the interaction continues. There are complicated implications brought about by the "shattering" of core memory resulting from continuing and varied demands. Therefore, a way of "clumping together" many small available contiguous segments of storage into larger segments is provided as an automatic feature of free storage. That is, each time a block is returned to free storage, an attempt is made to combine this block with other returned blocks to keep storage blocks as large as possible. This "garbage collecting" is the only system process which tends to counteract the normal shattering of free storage and is therefore very important.

As a safety measure, each block is made two words larger than requested. One of the words contains the size of the block and is compared to the block size when it is being returned. A mismatch is illegal and is assumed to be an error. The other word is used to string together all of the blocks for a particular console and provides the only means by which the system can recover storage given to an applica-

tion when an error occurs. This extra information is seldom destroyed by an undebugged application and can provide for full system recovery except in cases where only memory protection through hardware would suffice.

Interrupt Processors. The interrupt processor for the Digigraphics I/O channel is of special interest. It provides an analysis to determine the cause of display controller interrupts and branches to a sub-processor for further action. Types of interrupts that can occur on the Digigraphics channel are sector pulse, pen strike, maintenance, and keyboard.

Pen strike processing consists of enabling, at the controller, a sector pulse interrupt for the sector preceding the one on which the pen strike occurred and also inserting a mask in a table to indicate that, when the sector pulse interrupt occurs on the next drum revolution, an "identification" (ID) read operation is to be performed upon receiving the next pen strike. If light-pen tracking on a console is in progress, pen strikes from drum display are ignored temporarily. The resident maintenance interrupt processor checks for drum and/or controller parity errors that arise during drum I/O or information exchange with the controller.

Parity errors generally result in a second attempt before giving up. The keyboard processor results in a stacking of the console number and button number for future processing as mentioned in MAIN. The sector pulse interrupt processor is driven by a table. Given the sector number, it refers to the table to find out what needs to be done on this sector. The table provides an index for an array of entry points to routines to perform the following functions: read ID bytes from the drum; perform a drum read or write operation; update the position of the tracking cross and cause it to be displayed from core; perform a core display; increment a drum-timed clock; or enable a pen strike interrupt. Reference 4 provides a detailed description of the hardware involved for further information.

Nonresident System

Certain system routines were made nonresident because their frequency of use was assumed to be low. The combined size of these routines represents approximately 7K of core.

Byte Stream Generators. The byte stream generators form a set of routines which are FORTRAN callable for the purpose of generating the display beam driv-

ing commands and certain identification information to be placed on the drum. To display a line, for instance, the byte stream generator for a line is called giving it the coordinates of the endpoints of the line. This routine generates the 12-bit bytes which drive the CRT beam to cause the line to be displayed. After it inserts the byte stream into a resident buffer, another routine is called to generate the identification bytes which follow in the buffer immediately after the beam driving commands. These identification bytes are ignored by the controller when performing the off-line display function. They can, however, be read back into core from the drum by the interrupt processors when a pen strike from the line occurs. The ID contains information that is somewhat arbitrary; at present it consists of a code number which indicates that the displayed element was drawn by the console operator, the track and sector on which the beam commands for the element reside, a pointer to the element definition block in core, and certain other information which is necessary in order to extract the bytes from the sector if the element is deleted.

Routines are also provided for displaying strings of alphanumeric characters from four different fonts with variable character spacing. The strings can be displayed with any of three different ID byte codes. One causes the character BCD code to be packed into an application buffer. Another causes a button number and console number to be stacked for processing by MAIN (this is the light-button feature). The final ID is null and causes pen strikes to be ignored.

Maintenance Power Checks. This routine was made nonresident under the assumption that power failures would be infrequent. It provides for processing interrupts received when console power changes. Application initiation or termination can take place at this time as a result of console power change.

Recursive String Processor. Because most high-level coding languages, such as FORTRAN IV, are not adapted to building, searching, and referencing list data structures, a recursive procedure, callable by FORTRAN, has been provided to perform the search function. STRING is given, as formal parameters, the following: a function name, a pointer to the beginning of an arbitrary string, an integer which specifies the word number of the string component in an arbitrary n -component element, and one variable parameter which will be passed on to

the function when it is called. *STRING* steps through the list following the string component from block to block. At each block, the function provided as a parameter is called, giving it the variable parameter and a pointer to the block. The function may perform any operation on the block after which it returns providing *STRING* with a Boolean *TRUE* if a step to the next block on the string is desired and *FALSE* if the string processing is to be discontinued. Because *STRING* is recursive, it is possible for the function parameter to also call *STRING* at any block for the purpose of processing a substring. In this way, FORTRAN coded routines can process list data structures. For each different component in a block, an assembly language routine must be called to fetch or store a value. Using these assembly language routines, FORTRAN coded routines can build and reference blocks on a list.

An entry to *STRING* is provided so that if the function wishes to delete the block that is currently being processed, *STRING* may be signaled to perform the deletion. A corresponding routine to make insertions is not provided at this time because ordered strings have not yet proved necessary. Block insertions at the beginning or end of a string may be made by calling a special routine for this purpose.

STRING is normally a nonresident routine called through *XEQT*. This means that if a substring is processed, at the end of the substring a normal return will result that would tend to "unload" *STRING*. The frequency bit can be turned on or off by a call to a system entry, *FREQ*, to set the frequency. *STRING*, therefore, sets its frequency "high" unless only one string is being processed. The frequency of the function is also set high until just before the last call. If the function returns to *STRING* with a *FALSE* value, *STRING* calls a system routine to *FLUSH* the function. This action will cause it to be "unloaded."

APPLICATION PROGRAM STRUCTURE

In order to appreciate good response from the system, application programs should be structured to complement the systems capability. As mentioned before, applications consist of many "small" subroutines structured keeping the following points in mind. (There seem to be no hard and fast rules and an individual programmer's good judgment necessarily plays an important role.)

Dynamic vs Static Arrays

Because of the "shattering" of storage mentioned previously, it seems obvious that, although the use of dynamically dimensioned arrays is a required capability, it is not desirable to "overdo" this capability. For instance, consider a FORTRAN subroutine which requires several dimensioned arrays whose sizes are relatively small compared to the size of the subroutine. Specifically, suppose the subroutine size is 200 core words and three dimensioned arrays, each of size 5, are required. If these arrays are dimensioned internal to the subroutine definition, space will be provided for them in the same storage block with the subroutine. This means that at execution time, one block of size 215 would suffice. If, on the other hand, the arrays were obtained from free storage by *XEQT*, the breaking-up of storage which results not only requires extra work on the part of the system but may also result in noncontiguous blocks thereby degrading storage integrity.

On the other hand, suppose the subroutine size is 400 words and three 200-word blocks are required as dimensioned arrays. In this case, if the arrays are dimensioned internally, 1,000 words of storage is required in a contiguous block in order to load the program. After a lengthy period of system operation, the 1,000-word block may not exist! However, it is obvious that a 400-word block has a better chance of being available—as do the three 200-word blocks. In this case, it seems that externally dimensioned dynamic arrays might be more desirable. As can be seen, there is some uncertainty involved in the structuring of a subprogram requiring dimensioned arrays. At a given time during the system evolution, it is certain that past history has an effect on the "goodness" of a particular choice.

Program State Variables and Common

All variables which must be used by a particular subroutine and cannot be passed on to this subroutine by a concatenation of calls must be assigned space in *Common*. For instance, suppose a line is to be defined. A subroutine would be loaded to initialize the definition of a line by obtaining a block of storage to be used for parameters that define the line and relate it to its two end points and other lines. A pointer to this block must be left in *Common* after the initialization until the console operator has defined the first point. The first point may not be defined "soon." However, main frame control cannot

be held up in the defining routine by a PAUSE. The defining routine must return to its caller—in this case MAIN—leaving in Common the results of its execution. When the console operator defines the first point on the line, another routine is called which uses the variables left in Common and adds the coordinates of the point to a point definition after which it returns. At some later time, the second point is defined and another sequence of routines is loaded to complete the definition and cause the line to be displayed. As mentioned before, the system defines Common to the loader such that it will be the same for a given console no matter what subprogram execution is required.

Calls to Nonresident Routines

All calls to nonresident subprograms are made via the execution interface routine described previously.

Calls to Resident Routines

Calls to free storage to obtain dynamic storage blocks and *FREQ* to set frequency are made direct. Calls to assembly language coded routines for storing and fetching block components are also made resident since they are as much a part of the resident data as are the data values themselves.

Subroutine Frequency

There are several instances where a subprogram must concern itself with frequency. One of these instances was mentioned earlier in the discussion of the recursive string processor. Another is when a routine is itself recursive. In this case, the routine must set its frequency high until just before the last exit. This requirement places an added burden on the programmer and should be satisfied in a better way. However, the use of recursive routines seems to be limited at present.

Although these seem to be the only instances when a subprogram must concern itself with frequency, consider the multiple entry routine each of whose entries will be executed shortly after initial loading. If one of the entries initializes or sets up parameters defined in the body of the routine, it must remain resident or the parameter values and initial state will be destroyed by reloading.

Of prime consideration, also, is the overhead cost of loading a routine that is called, for instance, in a FOR-STEP-UNTIL-DO statement in ALGOL

or a DO loop in FORTRAN. In cases where a second or higher call can occur, response to an application becomes noticeably slower when the routine is resident on drum and can be prohibitive when disc input is involved.

Program Linkage

Subprogram linkage is accomplished in three ways. One way is by a direct subroutine call. Another, and less familiar way, is by a mechanism known as the active function stack. Considering again the definition of a line, at the end of the initialization phase the entry to the subroutine which will provide for processing the first point is placed on the active function stack just prior to a return to MAIN. When the console operator defines the first point, the action taken by the operator causes an application subprogram to be executed. The sole function of this subprogram is to remove the entry on the top of the active function stack and initiate a call to it via XEQT. Therefore, through a combination of sharing Common and the linkage provided by the active function stack, subprograms act much like the runners of a relay race—the first passing on parameters in Common to the second before it is loaded. A third and final way is through the concept of a request function stack. This concept has not been implemented for application use in the first phase of the system but existed in the prototype. It provides a way of “chaining” together subroutines that do not require any external action on the part of a console operator between subroutine execution. The request function stack is a system stack in the sense that MAIN would test this stack for an entry after it finds there is nothing left to do but go idle. In the current system, the I/O stack is a system request function stack.

CONCLUSIONS

As can be seen, an interactive application is one which takes full advantage of the fast response characteristics of the system and does not retain program control for an extended period of time such as would result from a lengthy iteration scheme or a tape-bound process. In order that an application may successfully be implemented within the system environment, a careful study of the problem to be solved must be made in order to determine that portion which is inherently graphic and requires the type of graphic interaction that

has been discussed. Unless the application can be analyzed in this fashion, it generally proves better to resort to batch processing to obtain printed or plotted output.

Furthermore, the application programmer must concern himself with the setting of subroutine frequency and the flushing of subroutines after use if his application is to function efficiently. (As mentioned before, flushing is automatic on exit if frequency is low.)

It has become painfully obvious that most present day compilers—FORTRAN N, where N=II, III, IV, for example—cripple a programmer's freedom in dealing with list structures. It would seem that computer manufacturers should have recognized this fact after so many years and begun to provide the needed capabilities. There exist compilers that do provide some of the required features, but they have not been accepted on a large scale and therefore defeat attempts to retain compatibility and machine independence. The only bright lights on the horizon seem to be the AED⁶ system being developed at MIT and a recent proposal⁷ regarding ALGOL. However, the present demand is still not satisfied and is not likely to be for several years.

As can be seen by the comments regarding the stacking of I/O requests and by the fact that the I/O routines are terminated by an interrupt at end-of-operation, time-sharing (in the sense used here) means that while an I/O request is being performed by the system to service one application, another application can have use of main frame or Digigraphic channel time. Control is idle in MAIN only when no requested functions for any console exist (console operators are thinking, perhaps) or when all applications have I/O requests stacked and incomplete. It is possible that a "time-slicing" scheme may be devised in the future, but the "natural" time-slicing provided by calls to non-resident routines (implied I/O requests) and explicit application requests for I/O seem to make "system imposed" time-slicing unnecessary.

At present, all compilations and assemblies must be done off-line under a standard batch monitor. No provision is made for concurrent batch and graphic job execution as is the case in some other systems servicing a single graphic console application.^{8,9} This provision can best be provided, perhaps, with a second core bank—a feature that may be added later.

Storage requirements for the system involve only

that of the resident portion which uses slightly less than 10K. The remaining 22K is controlled by the free storage program which is callable by both system and application programs. As can be seen, storage allocation for program residency is not a loader function. It is possible that only one other group of systems programmers⁶ uses a similar scheme for storage allocation. Console Common requirements are determined at application initialization time when application subroutines are written onto peripheral storage. A block of the appropriate sized storage is obtained and a pointer to it saved for use when loading. A typical size for console Common might be provided by the numerical control application which requires 200₁₀ words for state variable storage. It may be worth noting that this application has over 260₁₀ entry points to non-resident subroutines.

Many improvements are planned for the system. The most important of these to be implemented soon will submerge the XEQT feature below the source level and make it an invisible feature of the system. Additional graphic debugging aids will be provided by late fall.

Some of the more obvious advantages and disadvantages of the system can be listed as follows.

Disadvantages

1. High load function overhead compared to overlay schemes.
2. Restricted program structure.
3. Little storage protection.
4. Programs must be compiled or assembled off-line.

Advantages

1. Monitoring of subprograms.
2. Storage optimization on a demand basis.
3. Virtually unlimited application program extension capability.
4. Transparency of display console to application programs.
5. Good debugging features.

SUMMARY

All application and most system subprogram residency is made a function of console operator demand. That is, a subprogram is resident only when it is required to perform some computation in response to a demand made by the operator. Upon

completion of the computation, the storage which it occupied during execution is promptly returned to the system for future use by the same or another application. Furthermore, all application programs are structured in a highly modular form such that an application consists of many small subroutines which are loaded only when called and are allowed to remain resident only during their period of active execution. Application programs therefore consist of sets of subroutines residing in a relocatable form on a peripheral random access high speed storage device. Associated with each application program (console) is a resident table in a form which is sufficient to allow the system to find subprograms when they are called and cause them to be loaded into an available block of storage just prior to transfer of control to them for execution. System subroutines are provided for interfacing applications with the display consoles and all display drum I/O is handled by the system.

REFERENCES

1. S. H. Chasen, "The Introduction of Man-Computer Graphics into the Aerospace Industry," *FJCC*, vol. 27, pt. 1, Spartan Books, Washington, D.C., 1965.
2. I. E. Sutherland, "Sketchpad, A Man-Machine Graphical Communication System," *SJCC*, vol. 23, Spartan Books, Washington, D.C., 1963.
3. T. E. Johnson, "Sketchpad III, A Computer Program for Drawing in Three Dimensions," *ibid.*
4. —, "Digigraphic System 270; System Information Manual," Control Data Corporation Digigraphics Laboratories, Burlington, Mass., 1965.
5. D. T. Ross, "An Algorithmic Theory of Language," ESL-TM-156, MIT Electronic Systems Laboratory, Cambridge, Mass. (1962).
6. —, "AED-O Programming Manual—Preliminary Release #1," MIT Electronic Systems Laboratory, Cambridge, Mass. (1964).
7. N. Wirth and C. A. R. Hoare, "A Contribution to the Development of ALGOL," *Communications of the ACM*, vol. 9, no. 6 (June 1966).
8. E. L. Jacks, "A Laboratory for the Study of Graphical Man-Machine Communication," *FJCC*, vol. 26, pt. 1, Spartan Books, Washington, D.C., 1964.
9. C. A. Lang, "New B-Core System for Programming the ESL Display Console," MAC-M-216, Project MAC, Cambridge, Mass. (Apr. 1965).

THE LINCOLN WAND

Lawrence G. Roberts

Lincoln Laboratory, Massachusetts Institute of Technology
Lexington, Massachusetts*

An ultrasonic position-sensing device has been designed which will allow a computer to determine periodically the x , y , and z coordinates of the tip of a pen-sized wand. The device can replace the light-pen and RAND Tablet¹ for 2-D work, and extend the usefulness of such devices by virtue of the extra dimension available. The extremely large working space in which the WAND can operate allows it to be used for an entirely new set of pointing functions not directly connected with a display as well as the normal display control functions.

The specifications of the device as it exists on the TX-2 are as follows:

Working space	4 × 4 × 6 feet
Resolution02 inch—(12 bits/axis)
Sampling rate	25 cps
Computer time	1%
Absolute accuracy	0.2 inches

The technique currently being implemented uses four ultrasonic transmitters and one receiver. Each transmitter is pulsed periodically so as to produce a 20- μ sec burst of energy, bandpass limited between 20 kc and 100 kc. This burst arrives at the receiver after a time delay proportional to the distance between the two devices. The receiver amplifier is tuned for 50 kc and thus rejects most room noises.

* Operated with support from the Advanced Research Projects Agency.

Its output is clipped so that it outputs a pulse when the signal is received. This pulse is used to stop a counter which was started by the pulse to the transmitter. If any reflections are seen by the receiver they occur after the straight path reception and are therefore ignored. Ten milliseconds after one transmitter has been pulsed the next one is pulsed to find the distance between that transmitter and the receiver. In this way the four vector distances to the receiver are determined and thus its position in space can be calculated.

The major inherent advantage of the ultrasonic delay method of determining the position of a stylus is that the measurements are all delay measurements where a digital counter can provide a direct digital readout without requiring an analog to digital conversion. Sound is very convenient for measurement purposes since its propagation velocity is approximately one foot per millisecond. Thus, a 1-megacycle counting rate will resolve distance to .014 inch and a 13-bit counter will allow measurements of up to 9 feet. Allowing time for 11-foot reflections to die out, the transmitters can be pulsed at 10-millisecond intervals. Since four transducers are used, the total cycle takes about 40 milliseconds, providing an operating frequency of 25 cps. The hardware is currently arranged so that the computer is interrupted when a new count is completed, at which time it reads the counter value and the transmitter number (see Fig. 1). It is left to software to

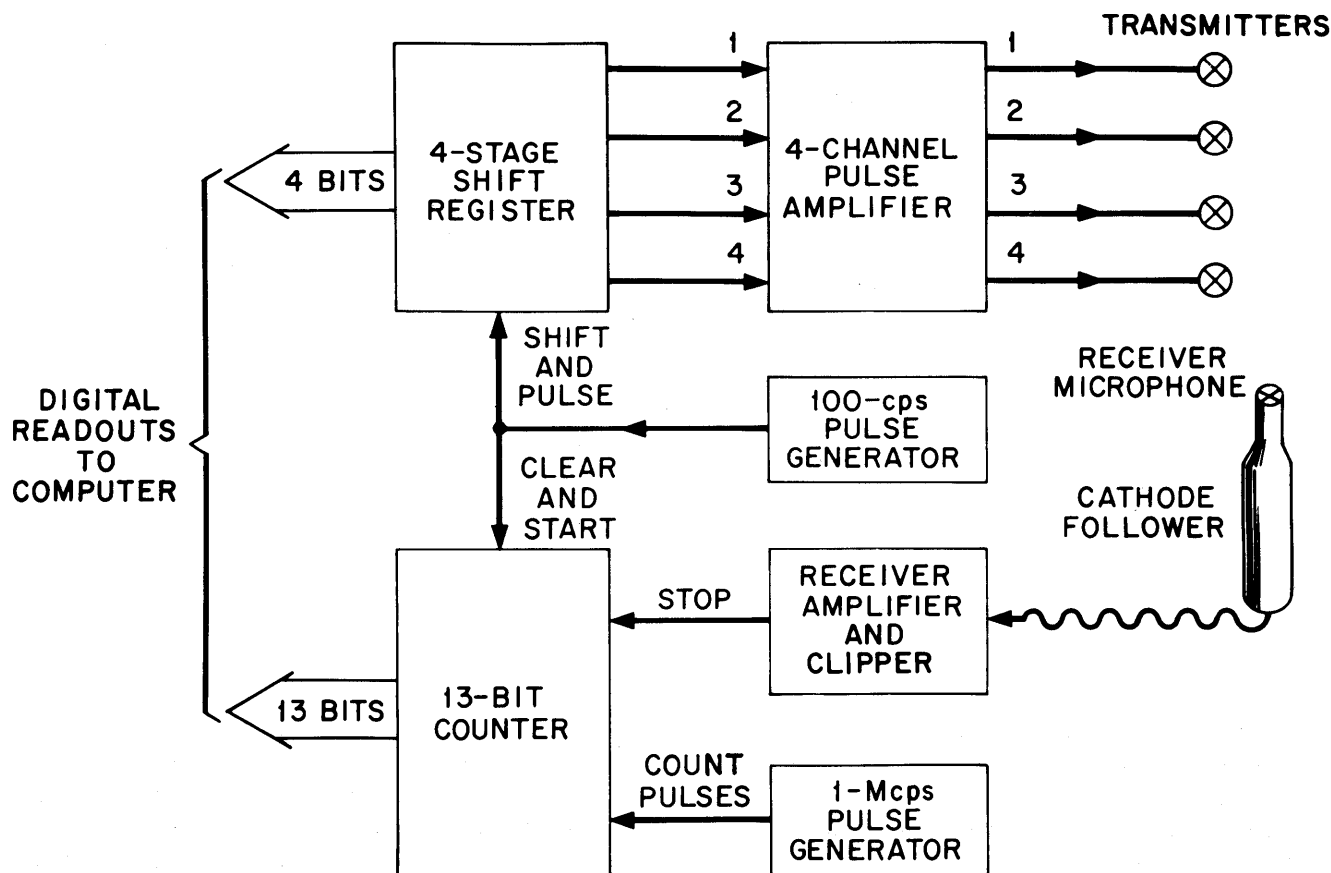


Figure 1. Block schematic of Lincoln WAND system.

calculate the x , y , z coordinates from the four distances.

DISPLAY POINTING DEVICES

There are two basic kinds of pointing capability which have proven important in graphic activities utilizing a display scope.^{2, 3} The first, item pointing, provides the machine with the command "that"; the second, position pointing, essentially says "here." Item pointing allows one to select one item out of many displayed on the scope thus facilitating multiple-choice answering or text and drawing editing. The position-pointing capability permits the construction of drawings and the repositioning of symbols on the scope. Without this ability the coordinates of each item would have to be typed in, making drawing extremely inconvenient. Both capabilities can be provided either by an item-pointing device like the light-pen or position-pointing devices like the RAND Tablet and the WAND. An item-pointing device can be extended to provide position information by displaying a tracking cross which follows it;

however, the cost in computer time is typically about 5% per console. A position-pointing device can be used to provide item-pointing capability if a hardware or software comparison is made between the coordinates of all points displayed and the position provided by the device. Such a hardware comparator has been built in the TX-2 display generator. The computer merely sends the device position (x , y) to a sample and hold circuit in the scope and if any subsequent vector or point generated by the scope goes near that position, the analog comparison circuits generate a program interrupt so that the identity of the item being displayed can be recorded. With comparator hardware for the central display generator, position-pointing devices are preferable to light-pens because there is no need to track them; the pointing is more precise, and they will work with long persistence phosphors.

POINTING IN THREE DIMENSIONS

The ability to provide conveniently three-dimensional position information makes it practical to

draw lines and curves in three dimensions.⁴ Translation of solid objects "stuck" to the WAND is also straightforward. In a slightly less direct manner rotations and viewpoints can be controlled. It is probably not profitable to use the three coordinates from the WAND for more complicated input data (such as rotation angles) unless the transformation is easily comprehended, because the user needs to be able to predict the effect of each movement using the 2-D display mainly for confirmation. An acceptable method of rotating an object would be to "fix" one point on it and move another point. This would control two angles of rotation and the object size. Alternatively, the distance between the fixed and moving point could be ignored to provide pure rotation. Usually, the ability to choose the points eliminates the need for a full three-angle rotation. A viewpoint for a new picture can be specified by first pointing out a new focal point and then specifying the location of the center of the focal plane. This technique provides the translation, perspective, and two rotation angles of a new viewing transformation. The third rotation angle is the extra one which rotates the picture on the scope face and normally would be fixed to keep the horizon horizontal.

In applications where a two-dimensional display is sufficient, the WAND can be used, in effect, as a large-scale RAND Tablet.¹ In this case, the z dimension is available as a scalar control. For example, consider the construction of a circuit diagram. There are two basic problems in such a process which usually require the use of a typewriter.

1. Each component has a numerical value which must be specified in some way to the computer.
2. Selections from a standard library of parts must be called up onto the screen whenever needed. Also various control buttons are required for specifying functions.

One method of obtaining numbers, parts, or functions which is very distracting is to put down the light-pen or tablet stylus, turn to the keyboard, and type the name or number. An alternative is to point to items around the edge of the display which can be filled with spare parts and control targets. However, there are usually so many parts and functions that several groups of items must be called up and searched to find the desired one. This "tree search"

is slow and loads down the edge of the scope with essentially dead pictures and labels.

The WAND provides solutions to both these problems because of its third control coordinate and the large working area. Numbers can be easily modified from their starting values by pointing at them and then "pulling" them up or down with the z axis control. The whole number could be adjusted or the mantissa and exponent separately. If the number were changed exponentially faster with increased velocity, a large dynamic range could be controlled. The large volume available for pointing can be used to point to target areas and pictures of library parts which are completely off the scope. Since the WAND has more range in x and y than the field of the scope and can operate in free space, it can be used to point at large boards of both photographs and typed sheets of function names.

The useful surface area surrounding a scope which would be accessible to the WAND is at least 5000 square inches. Even with one item per square inch this is an impressive library. The advantages of such a setup in reduced display load, preparation time, and specification time are obvious, but perhaps even more significant is the ease with which a user can simply point at what he wants. An even more ambitious step would be to include a 35mm slide projector with a ground glass screen and have the computer control the slide selection.

WAND SYSTEM GEOMETRY AND COMPUTATION

The physical layout of the transmitters determines the complexity of calculating the x , y , and z coordinates. However, an even more important consideration is the ease of computing an error check on new measurements before they are accepted.

When a distance is measured the error is usually very small or very large, thus a simple threshold limit on an error measure would eliminate most bad measurements. This is the reason for using four transmitters instead of three. Since any three distances determine the three-space position of the WAND, the fourth provides a geometric check on the measurements. Of course, some sort of dynamic consistency check could be used to reject wild measurements; however, when the WAND is jerked, such a check is almost sure to fail. Furthermore, one type of error, the reception of a reflection when the direct path is blocked, is very consistent when it occurs and can only be detected by a geometric check.

In order to provide a simple computation of the error check, the transmitters are placed at the corners of a square or rectangle. This arrangement also allows them to be mounted flush on the front panel of a display scope. Considering a square geometry, assume the origin of the x, y, z coordinate system to be at the center of the square and the transmitters to be at $x = \pm a$, and $z = 0$ (see Fig. 2). Since the square of the distance between two points is equal to the sum of the squares of the x, y , and z displacements, assuming the WAND is at (x, y, z) , the squares of the four measured distances are as follows:

$$d_1^2 = (x+a)^2 + (y-a)^2 + z^2 \quad (1a)$$

$$d_2^2 = (x-a)^2 + (y-a)^2 + z^2 \quad (1b)$$

$$d_3^2 = (x-a)^2 + (y+a)^2 + z^2 \quad (1c)$$

$$d_4^2 = (x+a)^2 + (y+a)^2 + z^2 \quad (1d)$$

The computations for x and y are extremely simple.

$$\begin{aligned} x &= (d_1^2 - d_2^2)/4a \\ &= (d_4^2 - d_3^2)/4a \end{aligned} \quad (2a)$$

$$\begin{aligned} y &= (d_4^2 - d_1^2)/4a \\ &= (d_3^2 - d_2^2)/4a \end{aligned} \quad (2b)$$

A relative measure of the error in both x and y is then given simply by

$$E = d_1^2 - d_2^2 + d_3^2 - d_4^2 \quad (3)$$

The computation of z is more difficult. Since all the transmitters are in a plane, only z^2 can be found. However, it is only necessary to perform one Newton iteration to produce a new z from z^2 if we use the last value of z as an approximation.

$$z^2 = d_1^2 - (x+a)^2 - (y-a)^2 \quad (4)$$

$$z = \frac{1}{2} (z_0 + z^2/z_0) \quad (5)$$

where $z_0 =$ previous z .

COMPUTATION TECHNIQUE AND TIME REQUIREMENT

Since the computation of the WAND position must be done continually for each console, unless special hardware is provided, it is important to keep the computation time down. For this reason an incremental technique has been developed.

Given a new distance, d_1 , for channel 1, the change in d_1^2 between this time and the last time it was sampled (4 units ago) is given by

$$\Delta = d_1^2(t) - d_1^2(t-4) \quad (6)$$

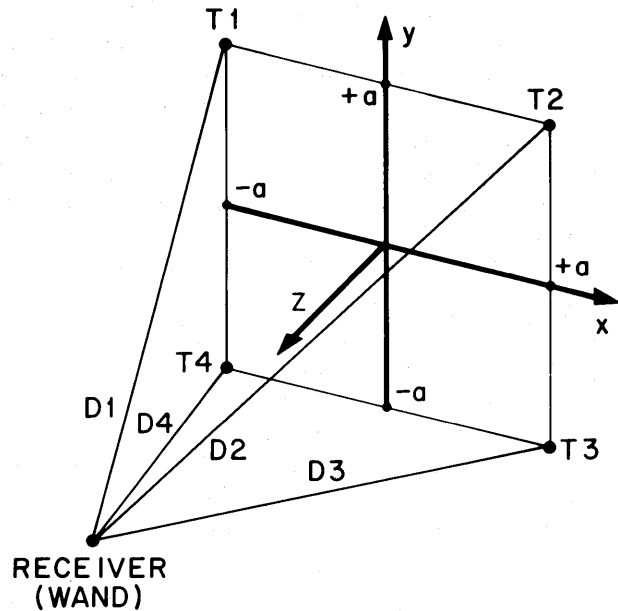


Figure 2. WAND geometry for receiver and four transmitters.

Then in order to calculate the new value of E we use

$$\begin{aligned} E(t) &= \Delta + d_1^2(t-4) - d_2^2(t-3) \\ &\quad + d_3^2(t-2) - d_4^2(t-1) \end{aligned} \quad (7)$$

or more generally,

$$\begin{aligned} E(t) &= E(t-1) \pm \Delta \\ &\text{(plus for } n = 1, 3 \text{ minus for } n = 2, 4). \end{aligned} \quad (8)$$

Providing we accept this error, we would save the new d_n^2 and E . Then every fourth time we would compute x, y , and z using Eqs. (9a), (9b), (9c), and (5).

$$x = (d_1^2 - d_2^2 - d_3^2 + d_4^2)/8a \quad (9a)$$

$$y = (-d_1^2 - d_2^2 + d_3^2 + d_4^2)/8a \quad (9b)$$

$$\begin{aligned} z^2 &= [16a^2 (d_1^2 + d_2^2 + d_3^2 + d_4^2 - 8a^2) \\ &\quad - x^2 - y^2] / 8a \end{aligned} \quad (9c)$$

These values of x, y , and z are computed from all (not just two) of the distances and thus average out errors better. The computation which is required each 10 milliseconds uses approximately 15 instructions on the TX-2 Computer and about 30 additional instructions every 40 milliseconds. Thus, the time required (on the TX-2) is about 1% of the total computer time.

ERROR ANALYSIS

Occasional errors arise in a gross way in the distance measurements because of room noise (mainly typewriters) triggering the receiver before the pulse is received. This would not occur if the capacitor microphones presently being used as transducers were replaced by more powerful transmitters. At present the error check eliminates a few samples each time a typewriter clanks. Errors also occur when one of the four direct paths is completely blocked. Then the receiver will either miss triggering altogether, which is easily detected, or one of the ever-present reflection paths triggers the receiver. This is the only case when too long a distance is observed. It is difficult to block the path with just a hand or arm unless one intentionally cups a hand over the transmitter. In any case these gross errors are easily rejected by the error check.

There are also fine errors due to the speed of sound changing in the air. A breeze is the most important cause of error of this type and can cause errors from .02–.1 inches in a 3-foot distance. Temperature changes also cause small changes in the speed of sound (about 0.1% per degree). Slow temperature changes mainly affect the absolute accuracy, not the short term stability. In addition to these effects, the pulse width at 50 kc is equivalent to 0.1 inch so the received signal must be detected carefully. Altogether, the fine short-term errors can be as large as about .02 inches in each distance.

Since x and y are computed from the difference of squares of these distances, the error can be magnified. If the transmitters are on a 20" \times 20" square and the receiver is three feet away at $x = y = 0$, then we have unity gain on errors from each distance. That is, a .02" error on one distance would cause .02" error in x and y . However, at six feet away the errors are almost doubled in their effect on x and y .

All effects considered and performance measured, stability of the position is as bad as 0.1" on an instantaneous basis. However, the programs have been designed to include damping of the x , y , and z values. With the damping averaging over 0.1 seconds the stability is about .02" and the tracking of reasonable hand motions is excellent.

The absolute accuracy has not been measured since it is of little importance in man-machine-display work. It is likely to be about 0.2" in the present system but could probably be improved with additional effort.

PLANS

The Lincoln WAND has been operational since April 1966. In the present configuration, operation has been mainly with test programs to determine the causes of errors and to find remedies in both hardware and software. It is expected that this development will continue. Application programs are also being designed. The WAND operates within the time-sharing system and the expansion to four or five WANDS is being planned. It is possible for all the transmitters in the same room to share the same pulsing logic.

A large portion of the cost of the current WAND is the \$1,500 for the ultrasonic equipment which should be reduced considerably in the near future. There are additional costs for the counter, pulser logic and receiver amplifier, but it is apparent that the total cost of the WAND should be competitive with two-dimensional sensors.

REFERENCES

1. M. R. Davis and T. O. Ellis, "The Rand Tablet: A Man-Machine Graphical Communication Device," *AFIPS FJCC Conference Proceedings*, vol. 26, pt. 1, Spartan Books, Baltimore, 1964, pp. 325–31.
2. Ivan E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," *AFIPS SJCC Conference Proceedings*, vol. 23, Spartan Books, Baltimore, 1963, pp. 329–46.
3. Timothy E. Johnson, "Sketchpad III—A Computer Program for Drawing in Three Dimensions," *AFIPS SJCC Conference Proceedings*, *ibid*, pp. 347–53.
4. L. G. Roberts, "Machine Perception of Three-Dimensional Solids," MIT Lincoln Laboratory Technical Report No. 315 (May 22, 1963).

USING A GRAPHIC DATA-PROCESSING SYSTEM TO DESIGN ARTWORK FOR MANUFACTURING HYBRID INTEGRATED CIRCUITS

J. S. Koford,* P. R. Strickland, G. A. Sporzynski, and E. M. Hubacher

*IBM Components Division, East Fishkill Facility,
Hopewell Junction, New York*

INTRODUCTION

This paper will describe a computer program that utilizes a graphic data processing system to aid in the design of mask artwork for hybrid integrated circuit modules of the type used in IBM System/360 Data Processing Systems. The system includes a small digital computer connected to a large-screen buffered display equipped with a light pen. A draftsman uses the light pen to assemble a circuit schematic on the display screen; simultaneously, a description of the schematic is entered into the computer memory. Thereafter, the draftsman can use the light pen to layout detailed artwork for fabrication of the circuit mask, subject to automatic checking against the stored schematic. When the layout on the display screen is complete, the corresponding mask artwork will be drawn by the computer via its digitally-controlled plotter. The graphical manipulations on the display screen, the automatic checking operations, and the control of the digital plotter are all part of a FORTRAN program that employs graphical subroutines to communicate with the light pen, display, and plotter.

* Currently with the Fairchild Research & Development Laboratory, Palo Alto, Calif.

Although the system is still considered to be in the development stage, its performance has already demonstrated that computer-aided graphics can significantly reduce the time required to produce the circuit artwork, and that the amount of data processing required is within the capabilities of the small computer. It is not only desirable, but even imperative, that the total time required for a circuit design to be converted from a hand-sketched schematic to the finished working module be made as short as possible. During the development stages of the module, fast artwork turnaround time will speed up the "breadboarding" of circuits to assure desired circuit performance. During the manufacturing stages, fast artwork turnaround time is valuable because minor physical modifications of the circuit layout used on a module will often result in a higher yield of acceptable parts. Usually these simple changes become apparent only after extensive observation of the automatic mechanism of the production line, and detailed statistical analysis of data derived from the line. There are many stations in the production line, and a layout change which results in an improvement at one station may not necessarily be an improvement elsewhere. Therefore, continual minor revisions of the mask used to make the module are very desirable, providing such changes can be made easily

and at a cost that justifies the change. The system described herein provides fast turnaround time for the original mask artwork and permits later revisions to be made quickly and easily.

Other major advantages of computer-assisted circuit layout are the increased accuracy and ease of record-keeping made possible by such a technique. Since the original circuit schematic and the finished mask artwork do not look at all alike, it is rather easy for mistakes to occur when manual layout is used for mask artwork. When computer-assisted layout is used, checking can be performed to insure that the artwork always agrees with the original circuit schematic. Once the correct circuit descriptions have been stored in the computer, they can be recalled at any time, examined, and updated. These descriptions could be easily merged with other engineering records in a large central data file. Automatic record-keeping and file-management techniques could then be employed to simplify the problem of keeping, updating, and managing engineering records.

The system described in this paper has the advantage that it uses a small computer with a standard programming language. The general problem of using computers to aid the engineer in the design process is currently being investigated in many places. Very large programming systems have been developed that employ large conventional computers to automate significant portions of the large-system design process.¹ Graphically-oriented computer languages and processor have been developed that simplify the control of computer-driven plotters and numerically controlled machine tools.^{2,3} Others have investigated the use of somewhat less conventional equipment to provide the engineer with new tools for accomplishing his designs.⁴ An outstanding example of the application of unconventional input-output equipment to a mechanical design problem is the MIT "Sketchpad" project⁵ that uses the computer-driven display and light pen as the basic communication device.

In the specific area of integrated circuit design, very useful results have been obtained with the CADIC System, developed by a group of engineers and programmers working under an Air Force contract at the Norden Division of United Aircraft Corp.^{6,7} The group has developed and implemented an effective off-line system, using card input and digital-plotter output, for the computer-design of monolithic integrated circuits. The group is currently

programming an on-line procedure that employs a light pen and CRT display for circuit layout.* The system described in this paper was developed independent of the Norden effort; however, many of the basic display functions in the two systems will, no doubt, be very similar.

THE SYSTEM HARDWARE

The equipment upon which the module layout program is implemented is shown in Fig. 1. As indicated in the block diagram of Fig. 2, the system is built around an IBM 1620 Mod. II computer with a capacity of 60,000 digits in main core storage. Attached to this data processor are two IBM 1311 Disc Files, each with 2,000,000 digit capacity, upon which all of the system programs and data structures reside. Conventional input and output are provided through the console typewriter, the IBM 1443 Printer, and the IBM 1622 Card Read-Punch. Graphical communication with the computer is provided by a direct-view display console equipped with a light pen. The display console uses a shaped-beam display tube that has a circular screen, 19 inches in diameter. The display is regenerated independently of the computer by an internal core buffer that has sufficient capacity to display at one time 1,023 straight-line segments, or about 5,000 characters, or a mixture of both.** The console is also equipped with a function key system that, along with the light pen, can interrupt the data processor at any time to inform its program of operator action. Hard copy graphical output is provided by a 29-inch incremental drum plotter.

The basic disc-resident programming system is built around the monitor system supplied by the manufacturer. An assembly language and FORTRAN II are available to the user, along with supervisory loader routines. A set of graphical subroutines, enterable from FORTRAN programs, provide all communication between the main program and the graphical equipment.

THE SLT MODULE

The SLT integrated circuit module used in IBM System/360 computers is shown in Fig. 3. The

* The Norden off-line system was developed under Air Force Contract AF 33(615)-3544; the Norden on-line system is being developed under Air Force Contract AF 33(615)-1395.

** Some flicker occurs on the display tube when large numbers of characters or vectors are displayed.

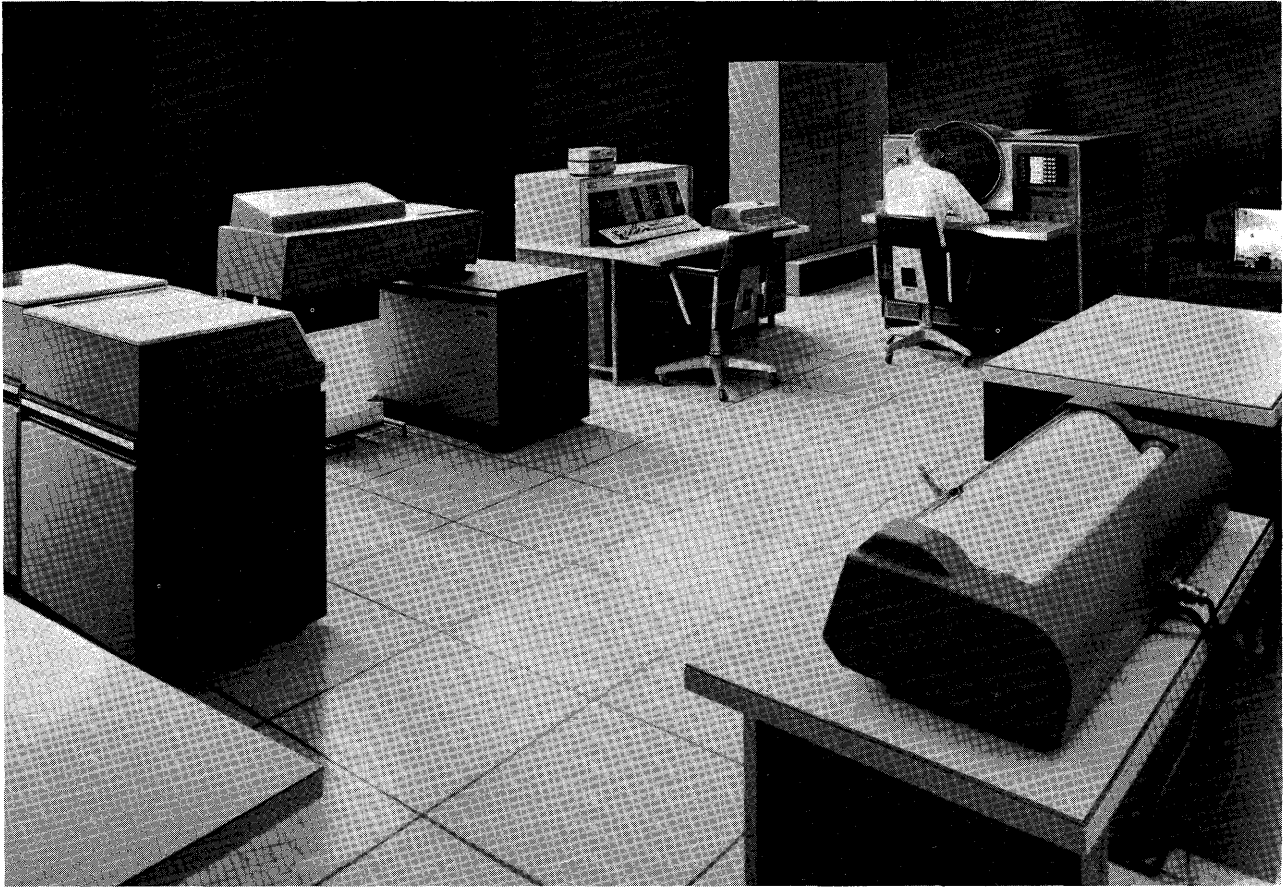


Figure 1. The graphic data processing system hardware.

module consists of a square ceramic substrate, 455 mils on a side, upon which electrode and resistor patterns are placed. Figure 4 illustrates the steps in the manufacture of such a module. Metal masks or screens are used to transfer the electrode patterns to both sides of the ceramic substrate, and also to apply resistor paste to the substrate. In subsequent steps

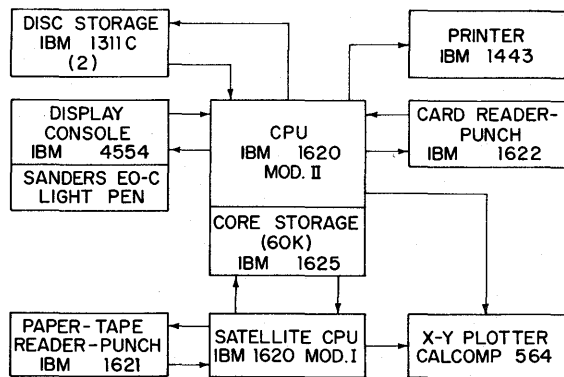


Figure 2. Block diagram of system hardware.

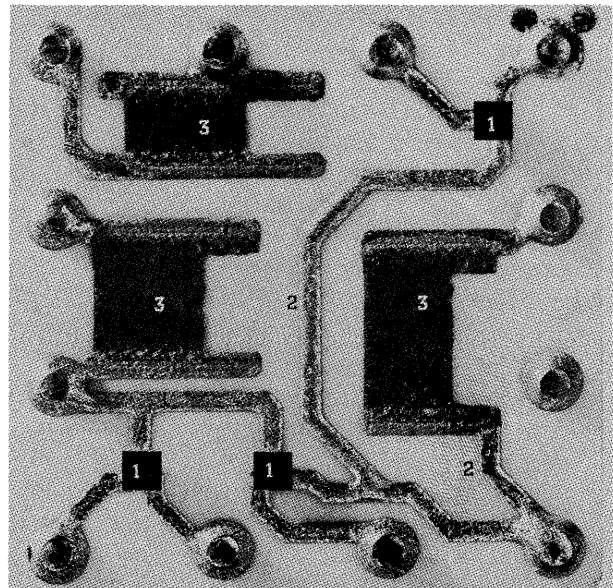


Figure 3. The SLT hybrid integrated circuit module; (1) chip transistors and diodes, (2) circuit pattern, (3) resistors.

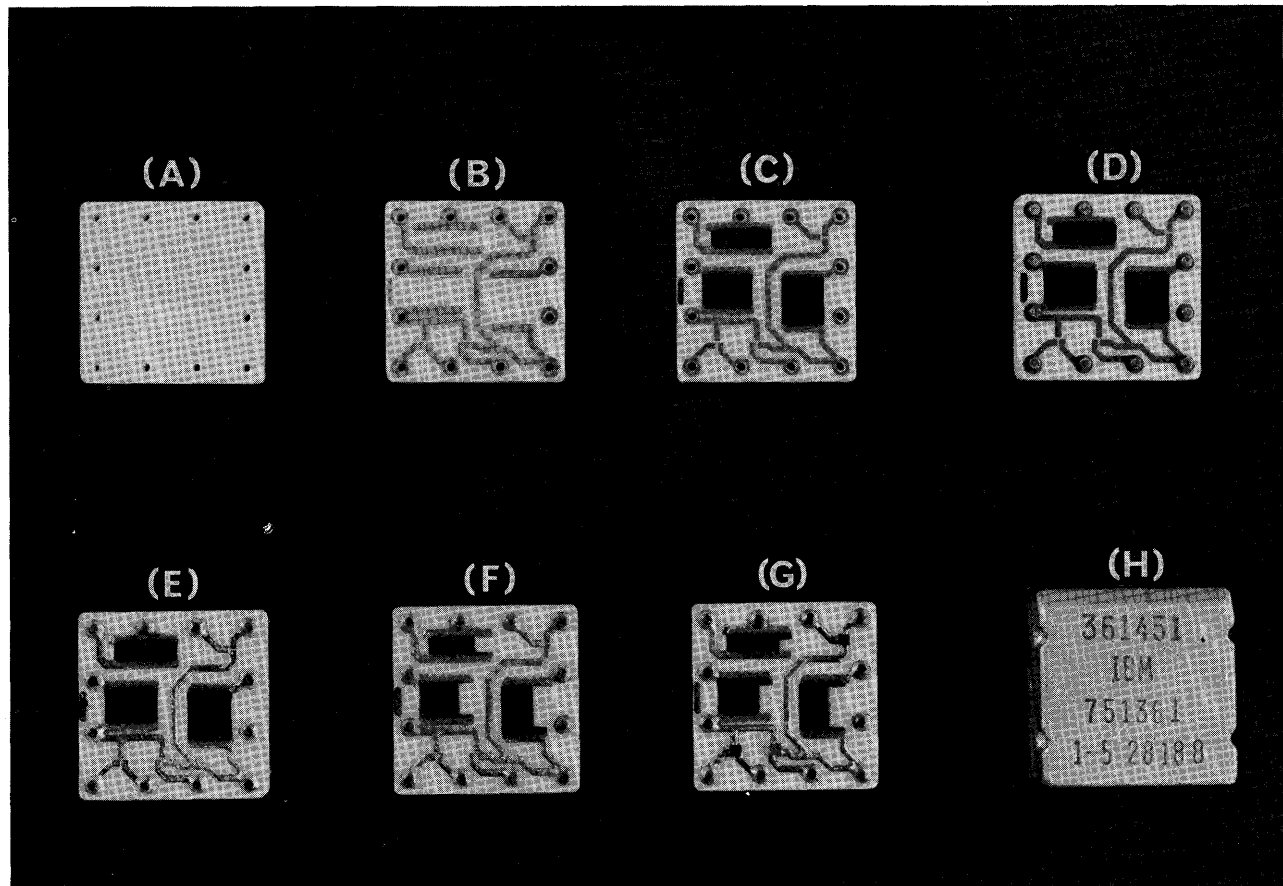


Figure 4. Steps in SLT module fabrication; (A) bare substrate, (B) print circuit pattern, (C) print resistors, (D) insert connecting pins in holes, (E) dipped into solder, (F) trim resistors, (G) attach chips, (H) encapsulate in metal shell.

the resistor paste is fired, the module is dipped in solder to tin the electrode pattern, and connecting pins are inserted in the holes of the module. Transistor and diode chips are next joined to the module, and finally the circuitry is encapsulated in a metal cap.

Figure 5 shows typical metal masks of an electrode pattern and its associated resistor pattern. The graphic data processing system produces artwork in the form of high-quality India-inked drawings on mylar tracing paper. These drawings can be transferred by photographic techniques to glass masters for making the masks.

A DETAILED DESCRIPTION OF THE OPERATION OF THE MODULE LAYOUT PROGRAM

Figure 6 indicates the general flow of the module layout program, beginning with the insertion of the

circuit schematic, and ending with the drawings of the artwork on the plotter. The draftsman begins by assembling a schematic diagram of the circuit on the display screen. A schematic symbol of any one of several standard components can be caused to appear on the display screen by firing the light pen on the appropriate "light button" at the bottom of the screen, as shown in Fig. 7. The draftsman then uses the light pen to assign a number to the component by either selecting a desired two-digit number, using a displayed matrix of digits, or by asking the computer to assign the next number in sequence for that type of component. A terminal of the component can then be connected to any point in the circuit by firing the light pen first on a circuit point and then on the terminal. As is shown in Fig. 8, a line will appear on the screen to indicate the connection. The draftsman can then place the component symbol in any position on the screen by moving a light pen tracking

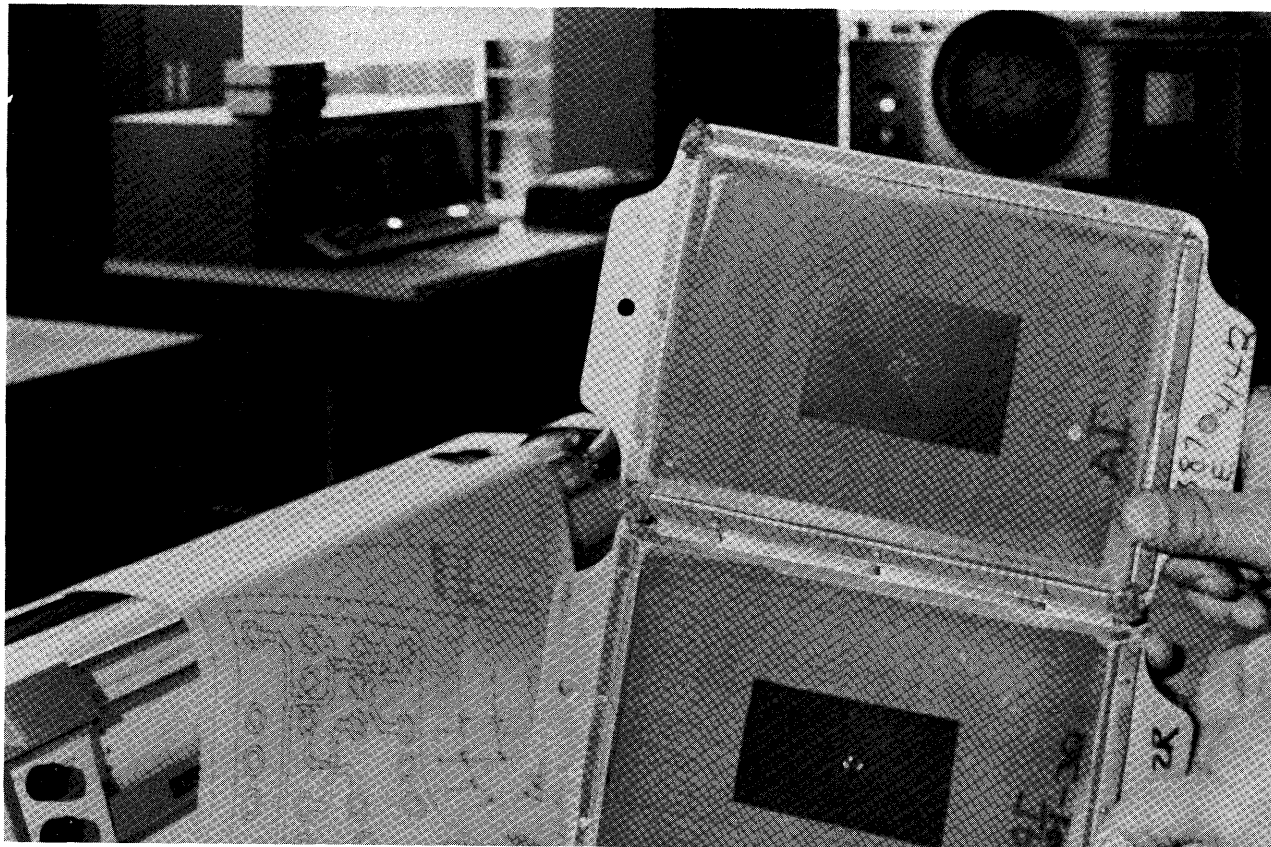


Figure 5. Metal masks for the electrode and resistor patterns on an SLT module.

pattern (the small square on the display in Fig. 8) close to the desired component position. The component will then be moved so that the connecting line can be drawn either horizontally or vertically to produce a circuit schematic that is pleasing to the eye, as is shown in Fig. 9.

There are several advantages in using the display-light pen system of input instead of punched-card input to enter the circuit schematic into the computer. Data are entered in a convenient pictorial format that enables easy checking for omitted connections or components. Since the program is self-teaching (i.e., at each step a message on the display screen directs the draftsman to the next step) the draftsman needs no programming experience and does not have to concern himself with input data formats. A draftsman with some experience can enter a typical circuit containing 25 elements and their interconnections in approximately 10 minutes.

Having assembled the schematic on the display screen, the draftsman can request via the light pen that the schematic either be drawn on the system

plotter, or that it be stored on punched cards. If the schematic is stored on cards, it can be re-entered at any later date from the cards for further modifications.

The next step in the design of the circuit layout is the entry via the console typewriter of the values and power dissipations of all the resistors in the circuit. After entering this data, the operator can specify the value of the resistor paste, or he can allow the computer to calculate the resistivity. This is done by firing the light button at an appropriate instruction on the display screen. The computer will then calculate the size of each resistor based on constraints of maximum power dissipation, tolerance for trimming, maximum electric field across the resistor, and minimum allowable length and width.

The most important part of the man-machine interaction occurs in the next portion of the program; i.e., the placement of the devices and the routing of the interconnecting leads on the module. Since the schematic has been entered into the computer, the information from the schematic is used to assemble

a parts list that is placed alongside a displayed blank module substrate, as shown in Fig. 10. At this point in the program, the draftsman is responsible for determining the placement of components on the

module and for overall layout organization. The computer is responsible for keeping track of the interconnections and insuring that the electrical characteristics of the circuit (as indicated by the sche-

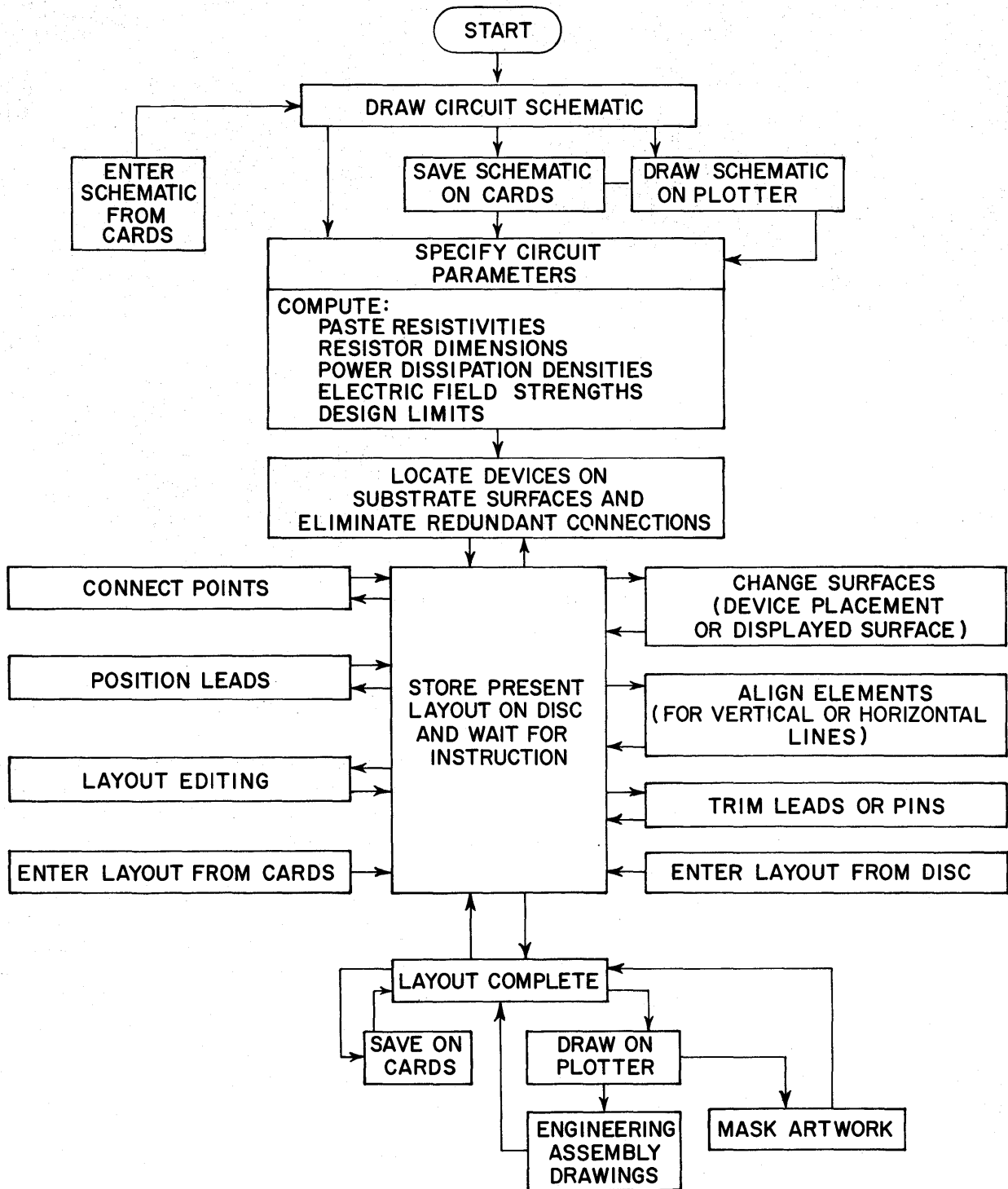


Figure 6. General flow chart for the module layout program.

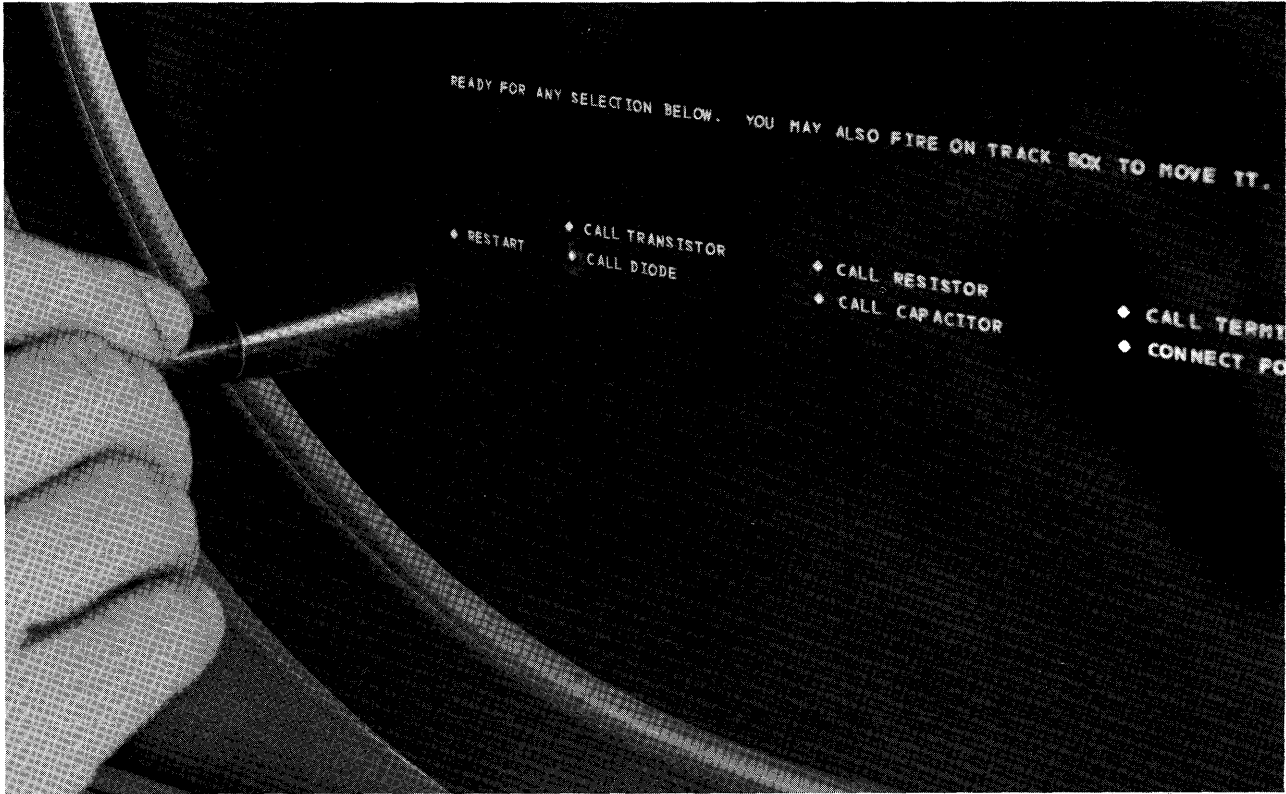


Figure 7. Labels on the CRT display screen permitting operator to select schematic components.

matic) are not altered by additions to or deletions from the circuit.

The draftsman employs the light buttons displayed at the bottom of the module representation, as shown in Fig. 10, to select programs for various options during layout. The first and most basic option is "LOCATE DEVICES." Having selected this option, the operator can place components on the module by firing the light pen on the labels in the parts list, and then moving the tracking pattern to the desired position of the devices. When a device is placed on the module, its label disappears from the parts list and a pictorial symbol appears on the display. These symbols are rectangles that are drawn to scale to indicate to the draftsman the exact area which will be occupied by the real devices on the actual module. Connections between devices are represented by single lines, and these connections appear automatically as the operator brings connected parts on to the displayed module substrate.

At any time, during or after placement of devices, the draftsman can move or rotate any component. All leads connected to the component are automatically repositioned and the electrical integrity of the circuit is preserved. Thus the operator can

very easily try a number of device placements to optimize some criterion such as number of cross-overs, lead length, or any other parameter. Figure 11 shows an operator working on a layout for the top surface in which some lead positioning and device placement has been done, and which has all of the components from the schematic placed upon the substrate. Three resistors from the parts list have been placed on the bottom surface, and therefore do not appear in Fig. 11.

Leads are routed by selecting the "POSITION LEAD" option. To route a particular lead, the operator fires the light pen on the displayed line representing the lead. A bending point is formed on the line, that can be moved to any position on the screen. The operator uses the light pen to move the tracking pattern to the desired position of the bending point. The draftsman indicates that the tracking pattern is at the desired position by firing the light pen at an "INACTIVE TRACK" message appearing on the display screen. The lead then will follow, much in the fashion of a rubber band. Up to 10 bending points can be formed in any one lead and the lead can be routed to any desired complexity. Figure 12 shows an example of lead-routing.

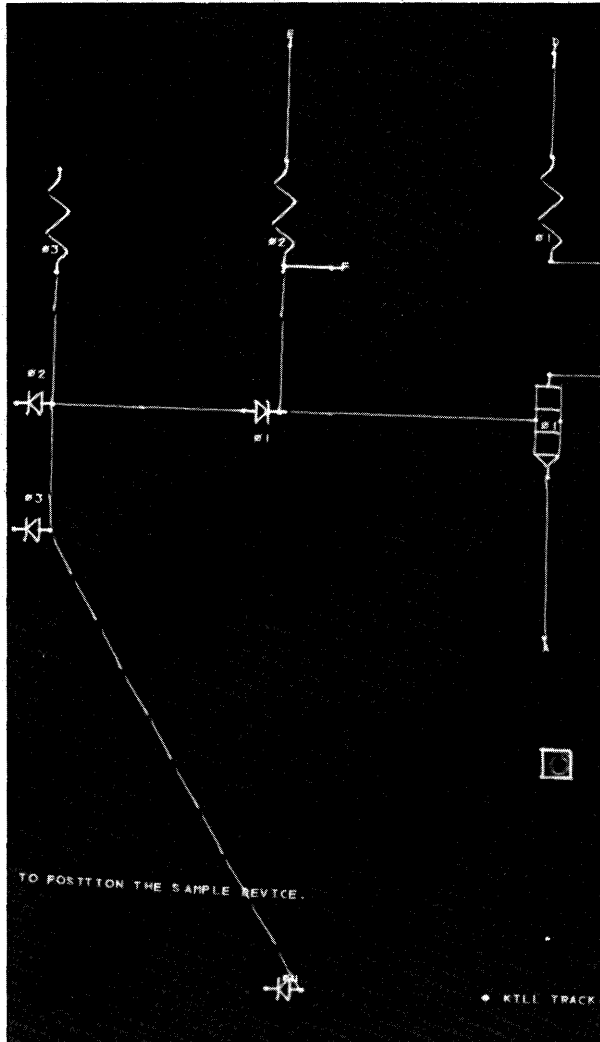


Figure 8. Connecting a schematic symbol into a partially-completed schematic.

To add new leads to the layout, the draftsman chooses the "CONNECT POINTS" option. He may then choose two points to be connected by firing the light pen at any two terminal points on the display, where a terminal point may be either a bending point of a lead or a terminal of a component. The program checks to insure that the requested connections does not violate the schematic; if no violation is found the connection is drawn upon the display screen. If a violation is found an error message appears on the screen and the connection is not established.

By returning to the "LOCATE DEVICE" section of the program, the draftsman may delete any connection from the layout, provided the deletion does not violate the circuit schematic. He selects a lead

for deletion by simply firing the light pen on that lead. If the deletion is valid, the connecting line will disappear; otherwise an error message will warn the operator of the violation and the lead will not be deleted.

Another option of the program is "TRIM LANDS." Around each module pin there is a ring-shaped land to which various leads connect. On a complex module where space is at a premium, some of these ring-shaped lands need to be trimmed. When the light pen is fired at the octagon representing a pin, a chord is constructed on the display which can be moved toward or away from the center of the octagon with the aid of the light pen. When the chord is in the proper position, the light pen is fired on the octagon and the section beyond the chord is trimmed away. Figure 13 shows the chord being used to trim pin 13 of the module.

In a similar manner the widths of interconnecting lands can be altered. In the displayed representation of the module, all connecting leads are represented by single lines. The draftsman can vary the width of a particular land by firing the light pen on the line representing that land. Lines indicating the outline of the land will appear along the representation line, and a land-width control pattern will also appear on the display screen. By firing the pen at appropriate points in the control pattern, the draftsman can shape the land in a variety of ways to fit the requirements of the layout. Figure 14 shows the display of a land

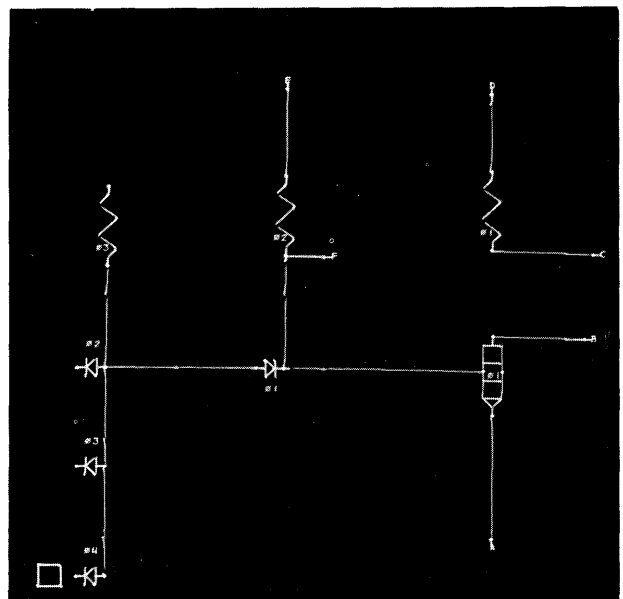


Figure 9. Schematic after component has been placed in position.

outline, with the land-width control pattern in the lower right corner of the screen.

The "LOCATE DEVICE" and "CONNECT POINTS" options have been subject to a checking feature which insures that the draftsman will not violate the original schematic. In the "RECONFIGURE" option of the program, a mode of operation is provided in which the draftsman can add new devices or leads, or delete old ones. While operating

in this mode, a blinking message on the display screen reminds the draftsman that the circuit-checking features are not operational. A very powerful feature of this editing program is the ability to call any one of 20 special devices from a previously-defined library of such devices. This feature allows non-standard devices to be called onto the module and manipulated as easily as standard devices. Hence, changes in the manufacturing process or in compo-

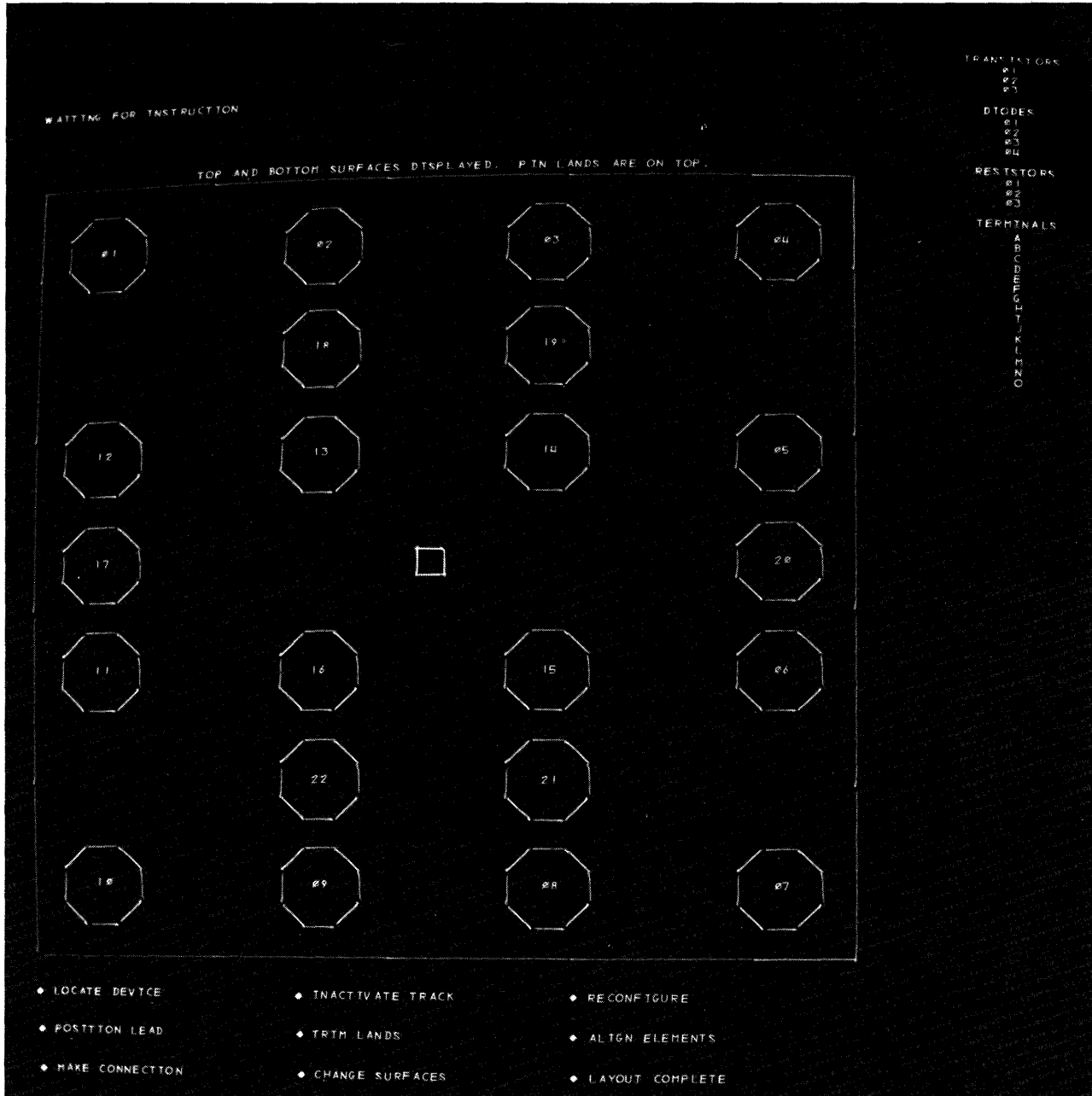


Figure 10. Blank module substrate and parts list as they appear on the CRT screen.

nent technology which alter component geometries do not require basic changes in the module layout program. Figure 15 shows a non-standard device selected and about to be connected into the circuit on a partially-completed module. Once devices are connected into the circuit by the "RECONFIGURE" section of the program, they will be recognized by all other sections of the program, and will be treated as if they had been entered in the original schematic.

In the SLT manufacturing process, resistors and connecting lands may be placed on either side of the bottom (pin side) of the module substrate. The "CHANGE SURFACES" option of the program allows the draftsman to lay out the top, bottom, or both sides of the module. The bottom of the module appears superimposed on the top, as though the

module substrate were transparent. All devices and lands are placed first on the top surface, after which the draftsman selects with the light pen the devices and lands that he wishes to place on the bottom surface of the module. Since the only connections between top and bottom surfaces are by the module pins, the program will not allow the draftsman to transfer a portion of the circuit to the bottom of the substrate unless it is connected to the top through a pin. The draftsman can lay out the top and bottom parts of the module simultaneously and can transfer components back and forth between the two sides. He may select either side of the substrate for display, or display both sides superimposed, with the devices and lands on the bottom side of the module blinking. A message along the top edge of the dis-



Figure 11. Operator working on partially-completed layout of the substrate top surface.

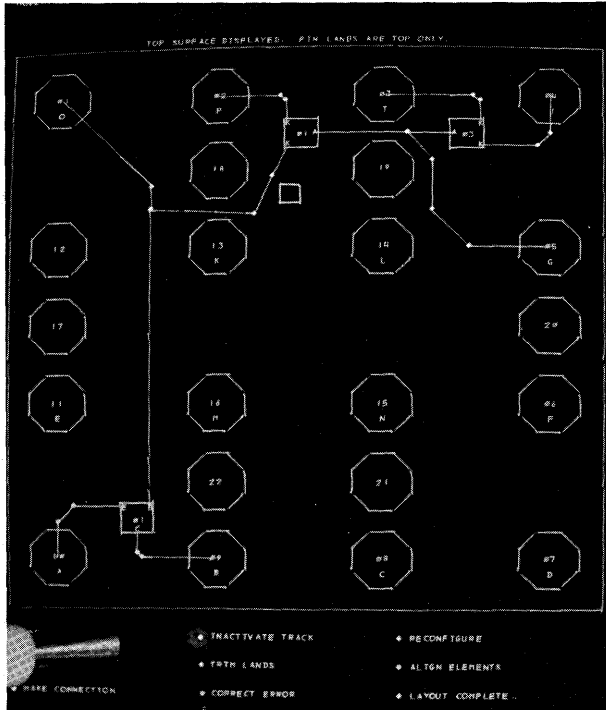


Figure 12. An example of lead-routing.

played substrate indicates which of the two sides is currently being displayed.

The "ALIGN ELEMENTS" option allows the draftsman to align all connecting lands in vertical, horizontal, or 45-degree orientations. Figures 16 and 17 show two versions of a module layout of the bottom surface containing the three resistors, one before aligning and one after aligning lands.

After the module layout is completed on the display screen, the draftsman can choose either to preserve the layout on punched cards (for subsequent re-entry) or to draw finished masks on the system plotter. If he requests that masks be drawn, a completely automatic plotting program converts the diagram of the layout on the display screen into finished drawings on the masks to any required scale. Up to four masks may be produced: two land masks for the top and bottom surfaces of the substrate, and two corresponding resistor masks. The artwork generated by the plotter is of sufficient quality that it may be photographically reduced to provide final-size masks for circuit fabrication. Figure 18 shows the four finished mask drawings for the land and resistor patterns of the module, Fig. 19 shows system-produced assembly drawings of the module shown in Fig. 18 for engineering documentation.

DISCUSSION

The module layout program has been implemented in FORTRAN on a small computer with limited main-memory capacity. It has, however, been possible to organize the program in such a manner that the speed and capacity of the computer are adequate for the graphical manipulations and associated processing. Most computer-initiated actions take place instantaneously as the draftsman requests them with the light pen. Actions which require involved list processing (such as checking for schematic violations) require more time, but rarely more than 15 seconds. All of the frequently requested actions, such as component placement and lead positioning, are accomplished within one or two seconds after the original request.

These processing speeds were achieved by keeping simple skeleton lists, containing summary information about the module layout, stored in the COMMON section of main-core storage. Detailed information, such as the exact routings of lands and the graphical configurations of the special devices, is read from the disc files whenever required. The skeleton lists contain, for the majority of the operations, sufficient information to allow them to be searched without reference to the disc file. In this manner, very simple list processing can be used to analyze light pen interrupts, with disc references only necessary when a required element is found in the list. When such an element is found, the data required can usually be read from the disc, updated in the computer and on the display, and rewritten on the disc within a few hundred milliseconds.

The disc is also used for core-image program overlays. Because the module layout program exceeds the amount of core storage available by an order of magnitude, it has been necessary to segment the program into discrete blocks, or links. A link and subroutines occupying the entire computer memory can be overlaid from the disc in about 3 seconds. In most cases the section of the program implementing one of the options described in the previous section will fit into a single link. Therefore, overlay usually occurs only when going from one option to another.

The FORTRAN-plus-subroutines approach employed with disc overlay techniques has proven to be an adequate method for implementing the module layout program. It has been possible to develop the required programs from scratch with an expenditure of about two man-years of effort. Furthermore, it

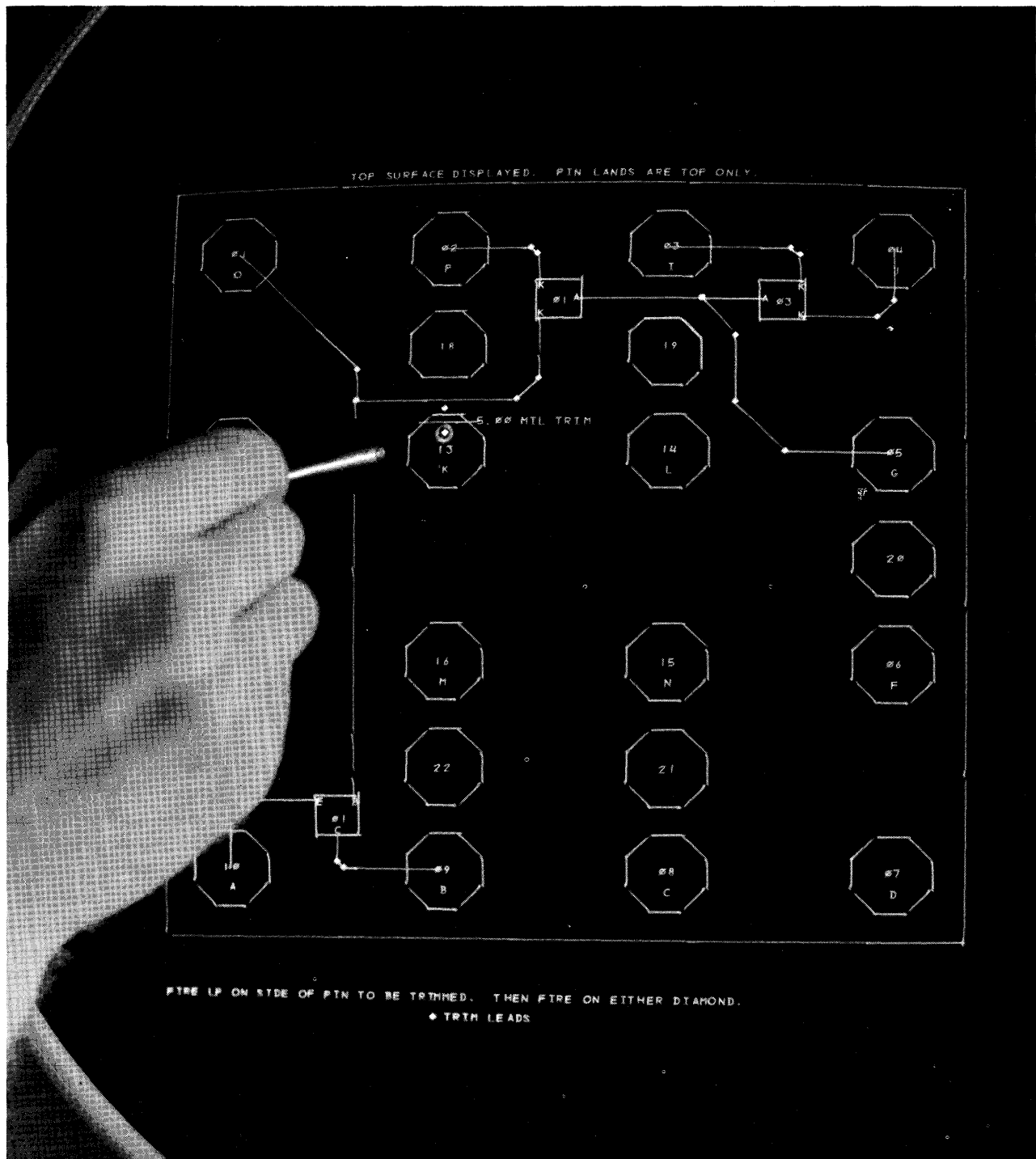


Figure 13. An example of pin-trimming.

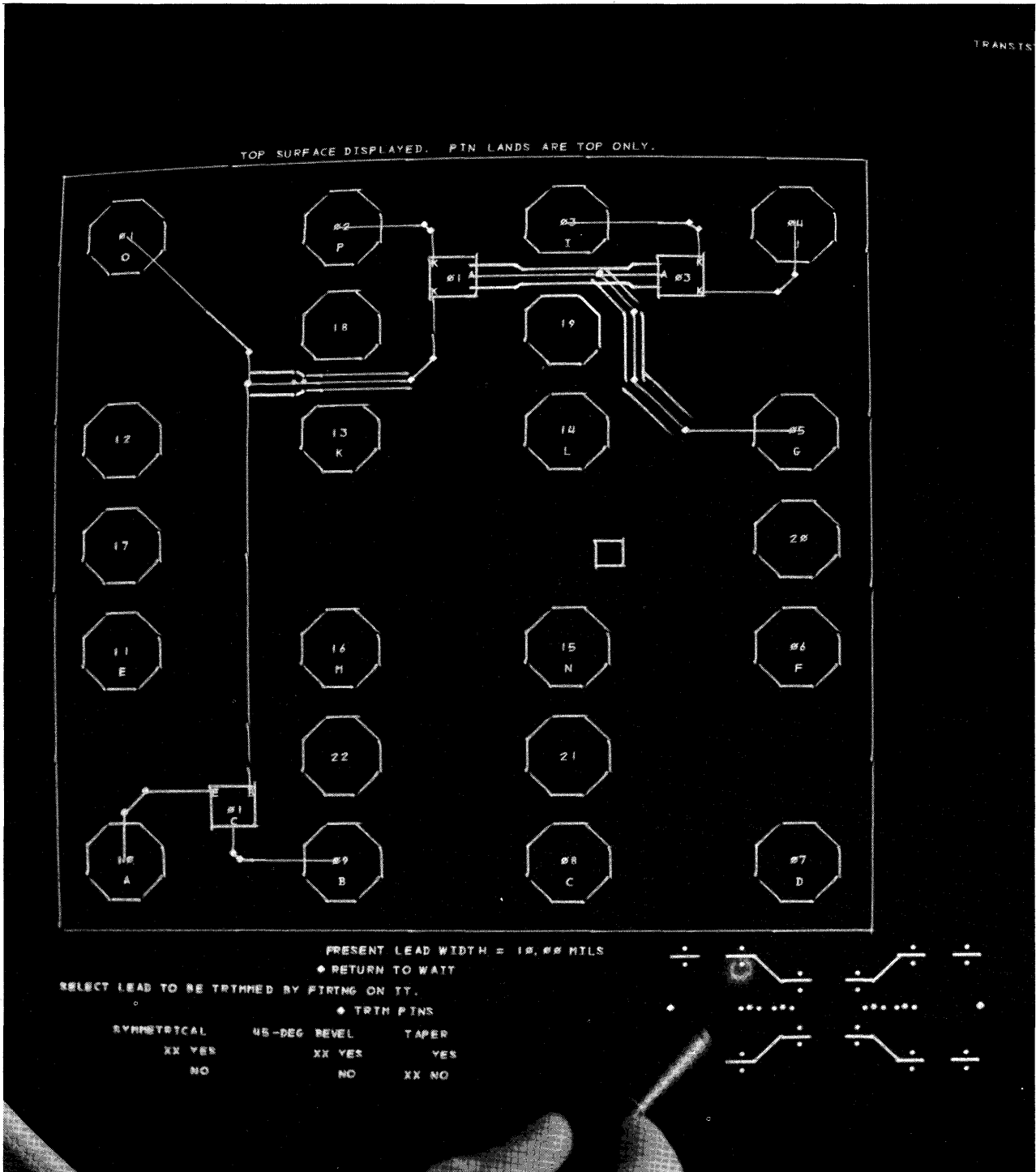


Figure 14. Display of a land outline with land-width control pattern in lower right corner of the screen.

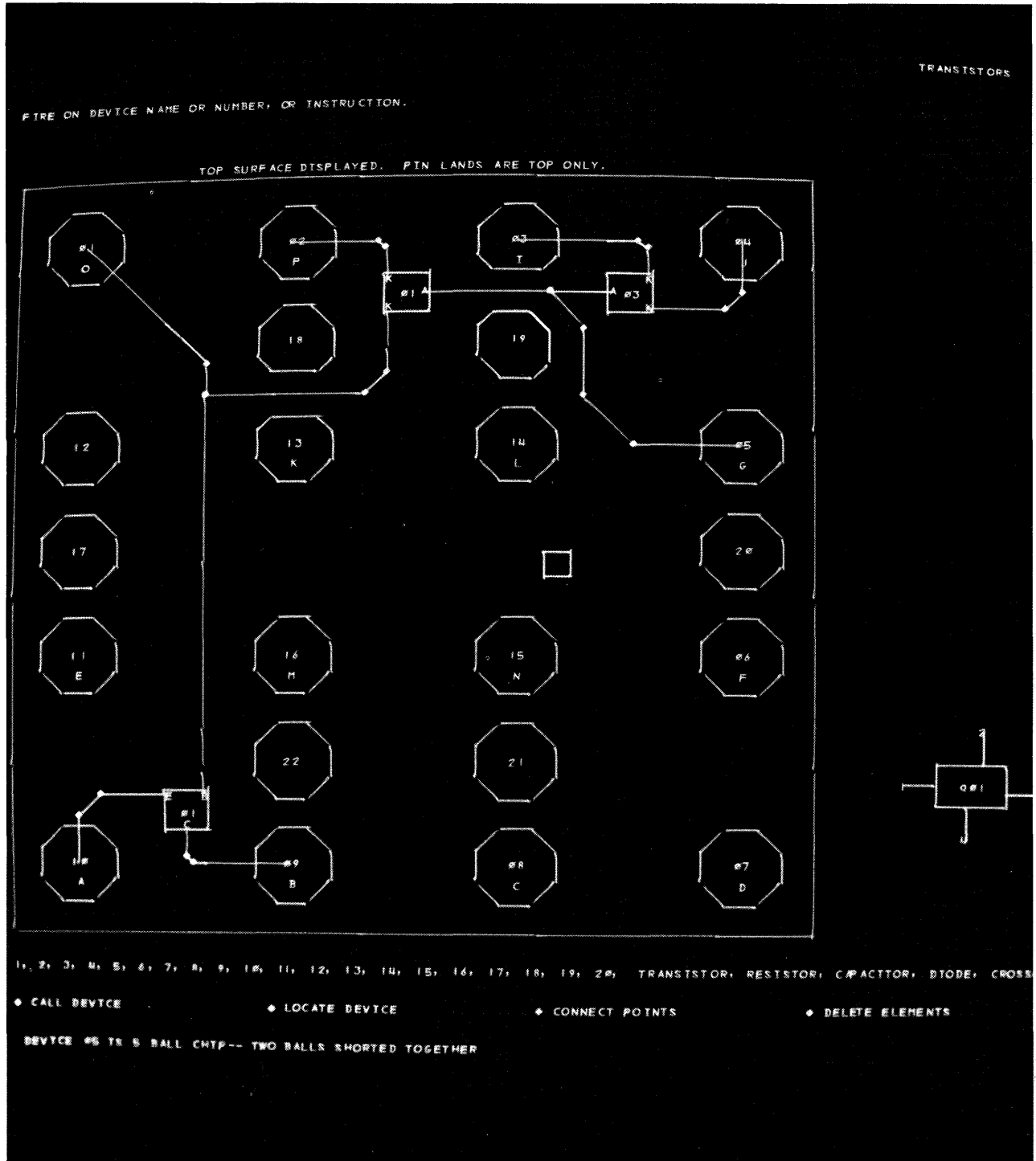


Figure 15. A non-standard device about to be connected into the circuit of a partially-completed module.

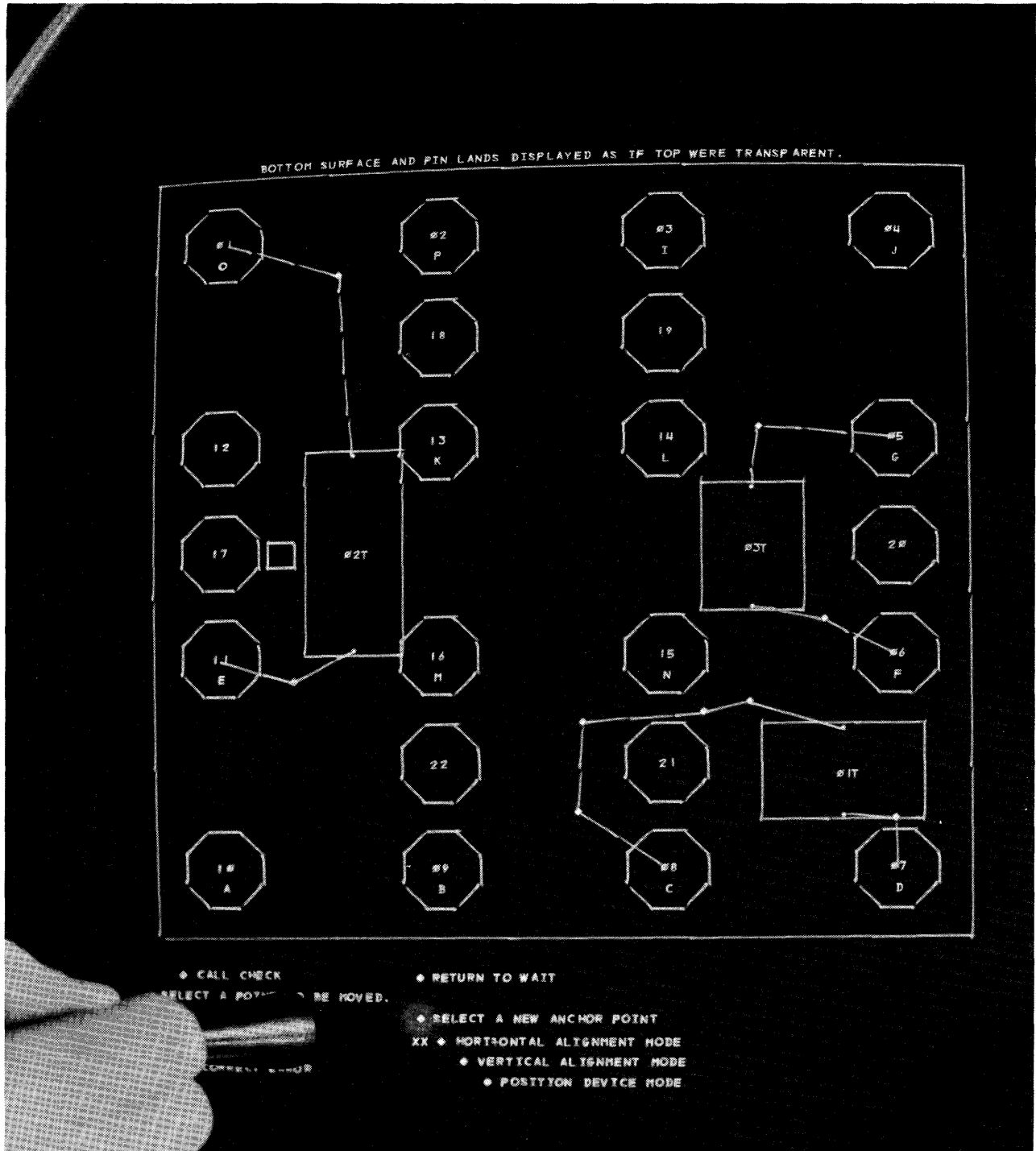


Figure 16. Bottom surface of module layout before land alignment.

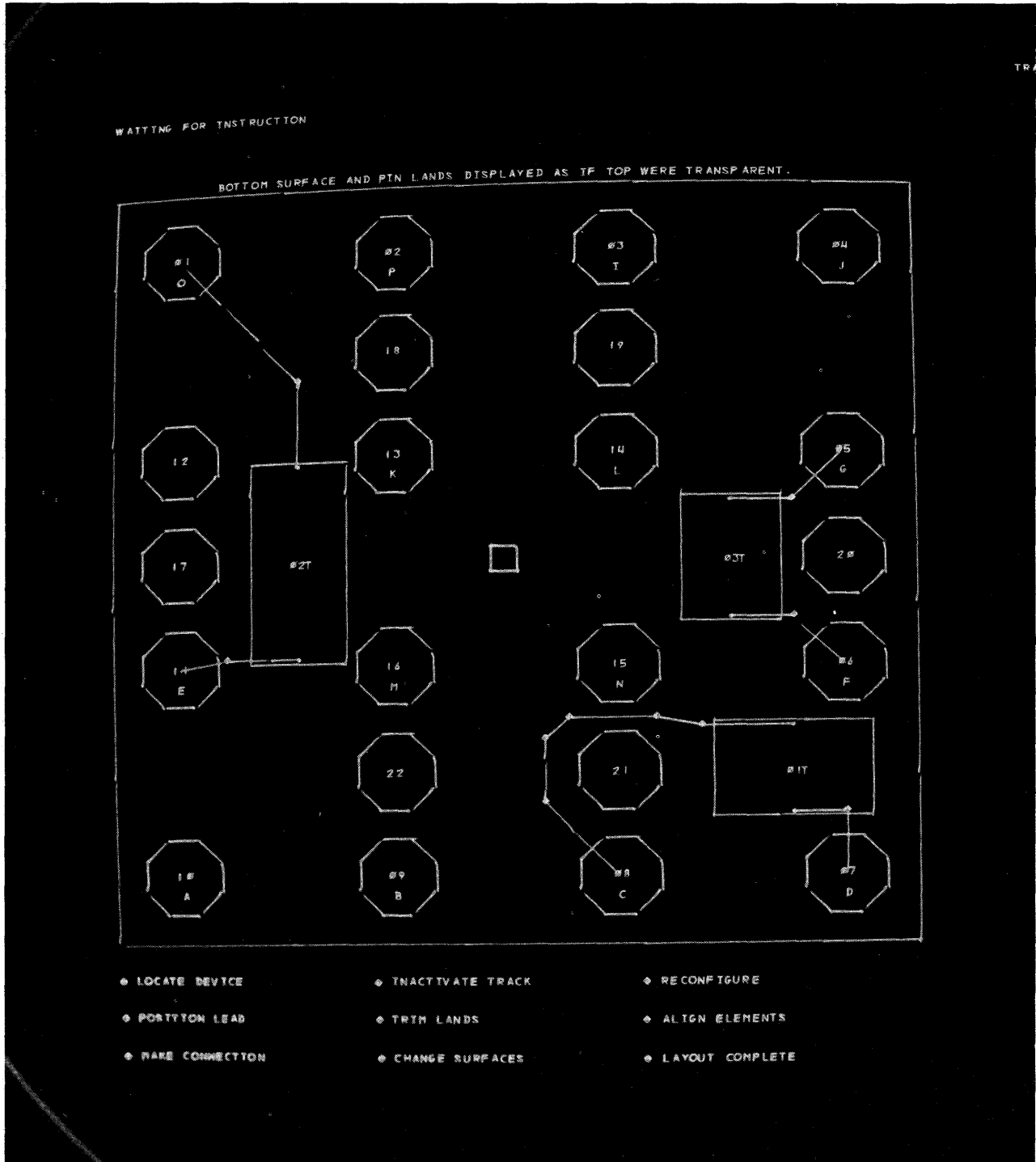


Figure 17. Bottom surface of module layout after land alignment.

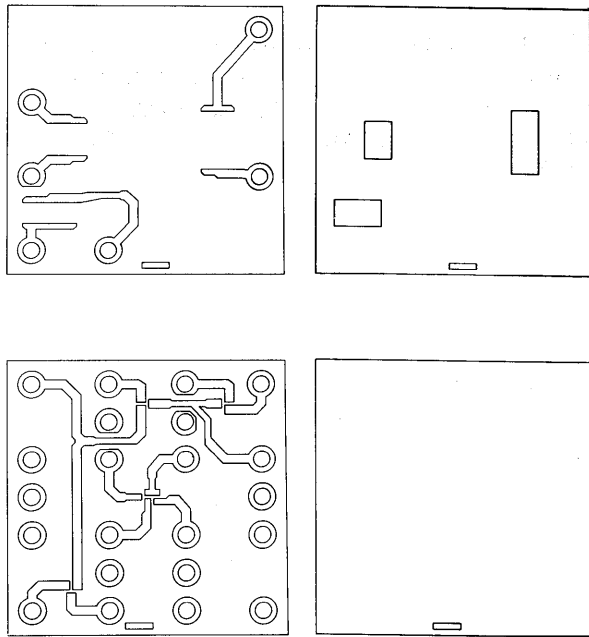


Figure 18. System-produced mask drawings for the land and resistor patterns of the finished module.

has been found to be relatively easy to teach FORTRAN and the graphical subroutines to individuals wishing to use the system. In addition, the use of FORTRAN has permitted easy modification of existing programs to improve their efficiency or to make them more convenient for the operator. Pending the development and widespread release of powerful graphical-programming and design languages,⁸ the approach used here has proven to be an extremely successful interim solution to the graphical-device programming problem.

CONCLUSION

The SLT module layout program represents a successful application of graphic data processing. Because the program is implemented with a standard programming language on a small computer, it appears to be quite feasible in the economic sense. While there remains much to be done, the development of the system described in this paper has shown beyond doubt that the use of graphic data processing techniques can result in significant improvements in both the ease and speed with which integrated circuit artwork may be produced.

REFERENCES

1. P. W. Case, et al, "Solid Logic Design Automation," IBM Journal, vol. 8, no. 2, pp. 127-140, (April, 1964).

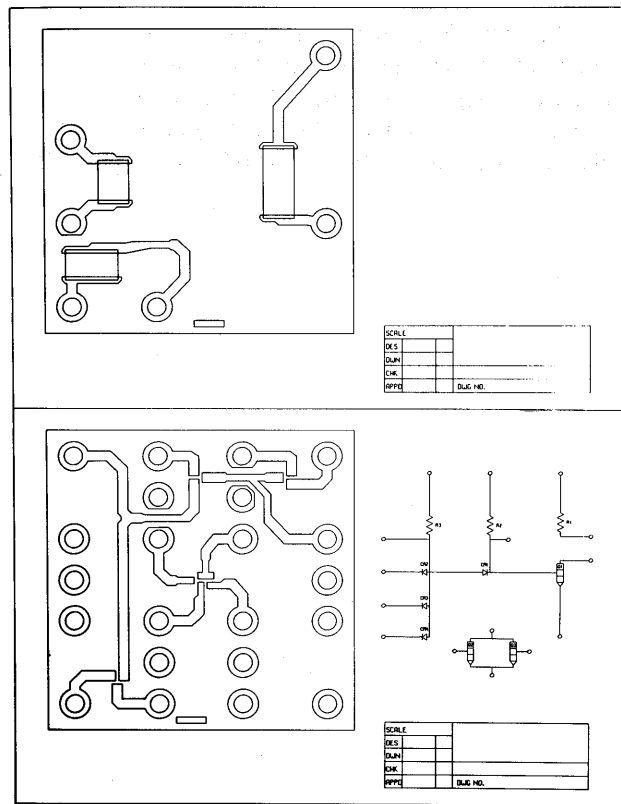


Figure 19. System-produced finished assembly drawings of the module in Figure 18 for engineering documentation.

2. Richard H. Waters, "A Drafting Document Generation Language," IBM Technical Report TR 02.328, San Jose, California, October 6, 1964 (parallels studies performed by Don Hollo at IBM, White Plains, N.Y., for the IBM 1620 System).

3. E. A. Bates, "Automatic Programming for Numerically-Controlled Machine Tools—APT-III," Proceedings of the 1961 Computer Applications Symposium, The Macmillan Company, New York, N.Y., 1961, pp. 140-156.

4. Donn B. Parker, "Solving Design Problems in Graphical Dialogue," University of California Extension Lecture Series on Man-Computer Systems, Lecture 13, Berkeley, Calif., Fall, 1965.

5. I. E. Sutherland, "Sketchpad, A Man-Machine Communication System," Proceedings of the Spring Joint Computer Conference, Detroit, Mich., May 21-23, 1963, pp. 329-346.

6. Final Report, "Techniques for Rapid Integrated Circuit Layout," Technical Documentary Report No. ML-TDR-64-103, Norden Division, United Aircraft Corporation, Norwalk, Connecticut, April, 1964.

7. Interim Engineering Progress Report, "Development of On-Line System for Computer-Aided Design of Integral Circuits," Norden Division, Project No. 9-521, United Aircraft Corporation, Norwalk, Connecticut, 1 March, 1966, 31 May, 1966.

8. D. T. Ross and J. E. Rodriguez, "Theoretical Foundations for the Computer-Aided Design System," Proceedings of the Spring Joint Computer Conference, Detroit, Mich., May 21-23, 1963, pp. 305-322.

AUTOMATED LOGIC DESIGN TECHNIQUES APPLICABLE TO INTEGRATED CIRCUITRY TECHNOLOGY

R. Waxman, M. T. McMahon, B. J. Crawford and A. B. DeAndrade

*IBM Components Division, East Fishkill Facility
Hopewell Junction, New York*

INTRODUCTION

Rapidly advancing integrated circuit technology has placed many new and often unforeseen demands on logic packaging techniques and, hence, is also impacting traditional computer design concepts. For instance, one of the most pertinent and immediate requirements is the optimum utilization of input/output (I/O) connections since the package size is strongly dependent on such connections. The package efficiency is measured in part by the I/O pin-to-circuit ratio, assuming the circuits in a package are connected in a way to provide an optimum logic function. Another potential problem to be considered is power dissipation, since integrated circuits may be contained in extremely small areas.

Consequently, it is imperative for a logic designer to interconnect many circuits within a package, thereby reducing I/O pins, yet not overburden a package with more heat than it can safely dissipate by the cooling provided. The above considerations are further impacted by the fact that integrated circuit technology inherently should provide inexpensive circuits. Thus the logic can be reasonably redundant to accomplish the minimum I/O pin-to-circuit objective as well as to minimize package types. The more versatile a package, the fewer types are required; the more functional a package (from

a non-iterative machine function point of view), the more part numbers are required. The most versatile package is a simple logic connective for example a NAND; however, the pin-to-circuit ratio is unacceptably high. A way must be found to obtain complex logic packages which do not require an infinite variety of package types.

A possible solution to the problem is to use Large Scale Integration (LSI), letting most packages be customized and still large enough to reduce the quantity of packages in any machine. The customization of each part leads to inventory problems at the place of manufacture. In addition, the complexity of each part may lead to many wiring layers which may be inaccessible for engineering changes. If an error is discovered during the fabrication process, the part may have to be scrapped. This delays delivery and increases cost of an acceptable part.

Another possible solution is the utilization of chips or cells having a certain defined logical complexity, flexibility, and acceptable pin-to-circuit ratio. This would enable engineering changes and modifications at the chip or cell level, thus making engineering changes of the interconnections between cells possible.

The proposed approach described in this paper is a compromise between a fixed interconnection pattern and entirely discretionary wiring between

cells on a wafer. That is, the interconnection of optimized fixed pattern logic arrays may be programmed to obtain various logic functions. Some related work has been done in this area. For instance, an array of logical elements which are interconnected in a specific pattern has been suggested.¹ Each logical element may be programmed to perform one of several two-variable functions by cutting certain interconnections within each cell. A similar approach has also been reported by AFCRL.² In that study it was proposed that each element be a NOR with possible interconnections to any or all eight neighbor NOR's. The interconnection pattern thus determines the function that is to be obtained. The work leads to speculation that a computerized approach might lend more sophistication to the synthesis procedure.

TRADITIONAL COMPUTER DESIGN PROCEDURE

The traditional procedure in designing a computer is first to define its major logical sections. The data flow paths are usually designed initially since they are generally well defined. Controls and non-iterative sections are determined last since they are heavily dependent on data flow organization. In recent years, the controls have become more organized through use of read-only memories (i.e., micro-programming). The read-only memory replaced much isolated and non-iterative hardware.

The various parts of the computer are still packaged in much the same way as they were first organized on paper. That is, the adders, registers, shifters, and other iterative networks are packaged to take advantage of their repetitive occurrence within the machine. The non-iterative sections are added without apparent organization.

How is the designer using integrated circuits presently? He is packaging iterative logical networks in an attempt to make efficient use of integrated circuits. These adders, registers, etc. have a high circuit density relative to the input/output connections to an integrated circuit element. He is packaging unit logic on integrated circuit elements to satisfy the non-iterative logical sections of a machine. These elements have a *low circuit density* relative to the input/output connections to an integrated circuit element. In other words, his design philosophy has not changed but succeeded in locking him into

a pattern that will not permit efficient use of monolithic technology.

How can the designer break out of this pattern? He is starting to change the pattern by packaging highly dense one-of-a-kind integrated circuits. This is efficient from the technology point of view, but creates inventory problems as well as greater fabrication cost due to the singularity of each part. Large Scale Integration (LSI) is being used in a manner which may reduce total parts, but not necessarily total part types. Problems of inventory and engineering change capability must be overcome. Also being investigated are the use of logical arrays in matrix form. Each cell at present is usually a simple logical connective, which results in an inefficient use of the elements because of interconnection limitations on the matrix.

To summarize, the relatively high cost of a component with discrete elements has never permitted the reduction of total part numbers by designing with a reasonable amount of redundancy. Although cost criteria and packaging techniques have changed today, logic design techniques have not kept pace. Economy now requires a reduction in parts, but not necessarily a reduction in logical elements. In fact, logical element redundancy can be a big factor in cost reduction if used properly. Efficient packaging today demands high circuit densities to take advantage of monolithic technology.

MULTIPURPOSE LOGIC CELLS AS A SOLUTION

An approach adding sophistication to the above ideas will be described in this study. Engineers already have taken the first step by proposing logical elements of the AND-OR variety (i.e., AND's feeding OR's). The limitation is that many a designer has been trained to see AND-OR groupings on the logic sheet, and that he imposes this AND-OR restriction on logic wherever permissible. He unfortunately is unable to see more complex functions that are not in the form (AND-OR), used in several equivalent forms. If he could, he would have available multiple-purpose logic blocks with an efficient circuit density relative to input/output ports. In an LSI application, he would have an array with complex yet versatile cells. Wiring of the final interconnection between cells could be one of the last steps of the process, perhaps allowing all personality wiring to be accessible for engineering change. Person-

ality wiring is the wiring of interconnections between cells which establishes the desired function of the LSI wafer.

A cellular array built on Large Scale Integration (LSI) principles would achieve the following objectives:

1. By making each cell a complex function, a large percentage of the wiring (the wiring of each element) could be completed before committing the package to its ultimate functional use.
2. By making each cell with a favorable pin-to-circuit ratio (where pin means I/O ports from each cell), the final personality wiring complexity may be reduced. This enables all personality wiring to be done on "outside" layers.
3. Reduce the inventory of different parts (since personality is the last step of the process).
4. Throughput to the user is faster since impersonalized arrays are available "off the shelf."
5. A mixture of fixed pattern wiring of cells with discretionary wiring between cells would result. This would provide a compromise between long computer-time high-wafer yield discretionary wiring between every circuit, and short computer-time low-wafer yield fixed-pattern wiring between every circuit.

The problem then is this: What logical functions should complex cell elements generate? This study describes a tool to aid in determining an optimum set of multipurpose logic blocks, personalized to a given computer or set of computers.

THEORY OF EQUIVALENCE CLASSES

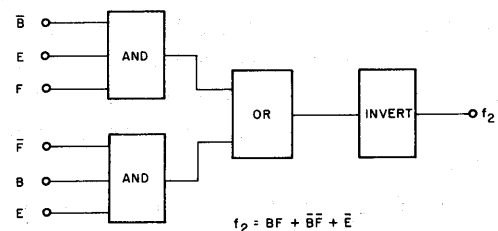
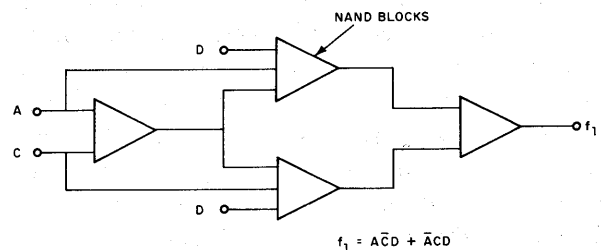
The basic theory to be applied to the automated design procedure proposed in this study is that of equivalence classes. Another approach might be that of pattern recognition of logic clusters. However, the pattern recognition approach lacks the ability to detect logically equivalent functions that have been laid out in different patterns.

Suppose a logic block existed which could implement a particular function with a given set of variables at its input. Suppose also that both the true and the complement of that function were available

at the output of this block. For the same input variables, two different functions are available at the output—the true and complement function. Now if the input variables are permuted (i.e., ordered differently on the input leads), it is possible to change both the output function and the complementary function into other functions which are, by definition, in the same equivalence class. That is, the internal circuitry of the block has not changed, but a different logical function has been obtained simply by permuting the input variables. If, in addition, both true and complement are available to the input (not both at the same time, but whichever one is needed), more functions both in true and complement form are available at the output. This last freedom of both true and complementary variables available at the input (one or the other, but not both) is reasonable since we are allowing both true and complement of every function to be available at the output of the logical block (see Fig. 1).

DESCRIPTOR PROGRAMS

The equivalence class descriptor program generates an identical octal number for any function in a particular equivalence class. The program is limited to functions of six variables or less. The input to the descriptor program is the octal number representation of the function column generated from a



IF $B = C$ THEN $f_2 = \bar{f}_1$
 $F = A \therefore$ THE TWO FUNCTIONS ARE IN THE SAME EQUIVALENCE CLASS
 $E = D$

Figure 1. Example of equivalence class theory.

6-variable truth table. Both the truth table and descriptor programs are written in FORTRAN and must be run separately due to the limited core storage capabilities of the 1620 computer. Once a logic cluster has been partitioned out, the computer running time to obtain the descriptor for the cluster is one to two minutes.

The output of the system is a list of 22-digit octal numbers which represent the output function for every cluster partitioned out of the original logic. By comparing the numbers in the list, it is possible to obtain the quantity of each equivalence class required to reimplement the original logic.

One of the major constraints of the system is its limitation of function size to six variables or less. With the present technique for generating a descriptor by manipulation of truth tables, it is not economical, time-wise, to handle larger functions. A 6-variable true table has 64 rows and 6 columns. For each additional variable, the size of the truth table matrix doubles in length and increases one column. In order to handle a 12-variable function, the truth table and descriptor programs would have to manipulate a matrix with 12 columns and 4096 rows. The octal descriptor would have 1366 digits. Even on a very large computer, the running time for descriptor generation and comparison would be in the order of hours for a 100 cluster partitioning. Also, a 1366 digit descriptor is not the ideal input to the logic designer who is asked to implement the function. Storage of a reference table of *all* functions with their descriptors is not practical since there are 2^{2^n} functions of n variables. To overcome these limitations, the following technique to generate an equivalence class type descriptor will be incorporated into the system. Descriptors for functions of greater than 12 variables can be realized. The descriptor is quite short and its length is not directly dependent on the number of variables in the function. There is also a correlation between the descriptor and the physical implementation of the logic (see Fig. 2).

The steps for obtaining the descriptor are as follows:

1. Calculate the Boolean expression for the output of the logic cluster that has been partitioned out of the original logic. The expression should be calculated in sum of products form. Either the true or inverse form of the output

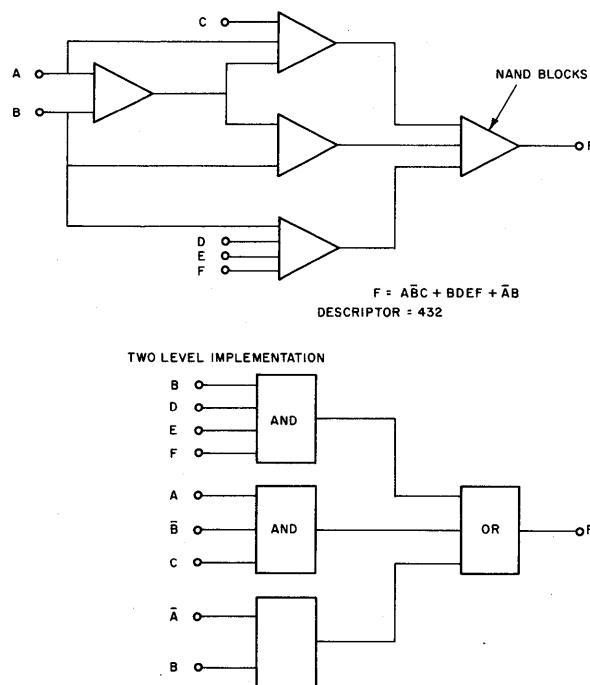


Figure 2. Descriptor example.

expression can be used, depending on which gives the sum of products form having the least number of terms.

2. Assign to each term in the expression a number corresponding to the number of variables in the term. An illustration is given below:

$$\begin{array}{rcl} \text{Function} & = & ABC + DEBF + AG \\ \text{Descriptor} & = & 3 \quad 4 \quad 2 \end{array}$$

3. Permute the digits to give the largest number; i.e., largest digit first, smallest digit last. The descriptor resulting from the function shown in the illustration will be 432.

Since we are allowing both true and complemented outputs of every function, the bars appearing over the variables in the Boolean expression do not have to be accounted for. If an inverted signal is required, the complemented output of the logic block generating the signal is used. In the case of the example given, the physical implementation of the function could be a 4-way AND, a 3-way AND and a 2-way AND, all connected to a 3-way OR block. Thus it is a very simple matter to go from the descriptor to an actual implementation.

It is possible that two functions in the same equivalence class (by the classical definition of equivalence) could have different descriptors. In actual practice, however, this would occur only a small fraction of the time. An important advantage of a descriptor of this type is that many equivalence classes can be implemented from one descriptor. To realize this advantage, the logic described by the descriptor would have to be implemented in two levels of logic. In the case of the descriptor 432, all functions having a descriptor of 3 digits, the first digit being 4 or less, the second digit being 3 or less and the third being 2 or less, could be realized by the logic used to implement the 432 example.

THE SYSTEM HARDWARE

An IBM 1620 Mod II Data Processing System coupled to a 4554 Video Display unit was used to implement the automated technique. The display has its own buffer, making it independent of the 1620 as far as maintaining the visual pattern on the CRT. The logic designer may communicate with the display unit by use of a light pen or through two sets of pushbutton switches.

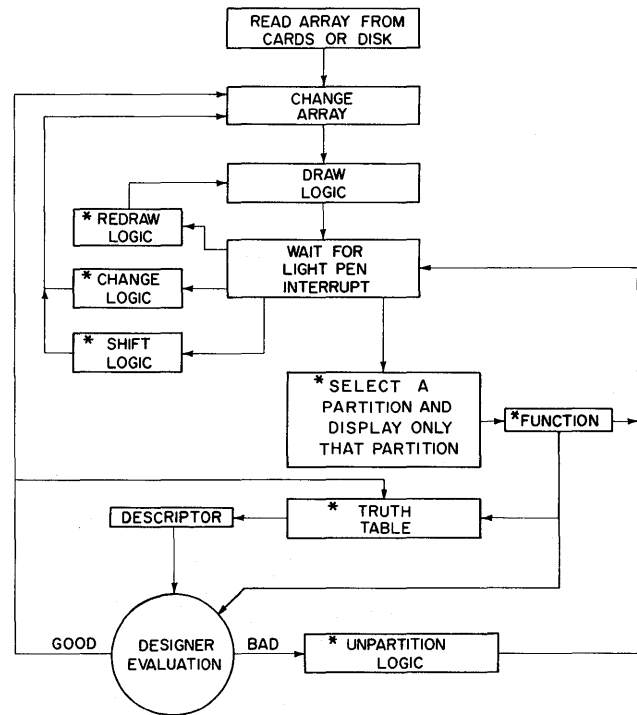
The program and data may be read into the 1620 from cards, or it may be stored on a disk pack and called by name into the 1620 core storage. Because of core memory limitations, all programs are stored on the disk and called into core storage by the main program when needed.

MAN-MACHINE PARTITIONING TECHNIQUE

The automated technique consists of a general man-machine interaction program to aid the designer in partitioning logic to a useful set of multipurpose logic blocks. It is in a form compatible with a logic designer's present partitioning method—that is, it allows a designer to cull out of larger sections of logic, small partitions that have meaning to him. The advantage of the automated synthesis technique is that it results in computer building blocks consistent with technology requirements of Large Scale Integration without mushrooming part numbers.

A typical application of the system will be described. Reference to Fig. 3 will aid in understanding the flow through the system. Figure 4 illustrates the IBM 1620 Mod. II system employed in the automated partitioning technique described in this section.

The first step in the procedure is to generate the



*INDICATES LIGHT PEN COMMANDS

Figure 3. Flow chart of logic partitioning.

logic preliminary to partitioning it. Initially, the program is ready for a light-pen interrupt. At this point the designer sees an array of 7×7 diamond shaped dots on the screen. At the bottom of this array is a set of instructions, as shown in Fig. 5. By firing the light pen at the touch point adjacent to each instruction, the designer may choose the desired action. In designing logic, the first step could be to activate the touchpoint adjacent to the type of logic block the designer wished to draw, for example "AND." Once this has been done, the light pen may be pointed to any of the diamonds in the 7×7 array. As each diamond in the array is addressed by the light pen, a logic block will be drawn around it and labeled as the particular logic function corresponding to the instruction touchpoint previously activated (see Fig. 6.). The logic on the screen may be designed by interspersing AND's, OR's and INVERTERS, or the designer may work from a pencil sketch and is able to put on the screen all the AND's followed by all the OR's, followed by all the INVERTERS, in any sequence desired.

Once all the logic blocks are on the screen, as shown in Fig. 7, the designer can point the light pen at the instruction entitled "DRAW LINES." He may



Figure 4. The IBM 1620 Mod. II Data Processing System, coupled to an IBM 4554 Video Display Unit, used to implement the automated logic partitioning technique.

then connect the logic blocks in the desired pattern by pointing to the end point of each line he wants drawn (see Fig. 8). The computer program then draws the lines. When the logic is completely interconnected (Fig. 9), the designer may then point to the instruction label entitled "LABELS." A box with labels of one letter will appear on the bottom of the screen. The initial letter in the box is "A." The designer may then point to each input he wishes to label "A." If he wishes to label an input "B," he points the light pen to the block with the "A." The label changes to "B" and all inputs subsequently addressed by the pen will be labeled "B" (see Fig. 10). He may then continue to label all inputs down through "Z" (Fig. 11).

Should the designer make an error in drawing the blocks, lines, or labels, there are instruction touchpoints for erasing blocks, lines, and labels. He may then go back and draw in the correction by the appropriate light-pen procedure.

In place of a manual input, the system is capable of accepting into the disc file up to several thousand blocks of interconnected logic. Up to 49 blocks of this stored logic may be placed on the screen at any one time.

When the logic has been placed on the screen, the designer may point at the instruction touchpoint

entitled "FUNCTION." When the program is in this mode, pointing at any block output will result in the Boolean function being printed out on the screen for up to three prior levels of logic; i.e., any labeled inputs up to three prior levels, as well as the designation numbers of any blocks feeding the third level of logic, will appear in the Boolean expression. The number of variables in the function mode of operation is not limited (Fig. 12).

If the designer desires to partition the logic into smaller segments, he may point his light pen at the instruction touchpoint labeled "PARTITION." Then, by pointing the light pen at the connected logic blocks which he desires to partition from the overall cluster on the screen, he directs the computer which blocks are to be partitioned. The next operation is to energize the instruction touchpoint labeled "DRAW PARTITION." All logic then disappears from the screen, except for the blocks that were partitioned out, as shown in Fig. 13. If the function is limited to six input variables, the touchpoint labeled "TRUTH TABLE" may be energized.

When the output for the point at which the truth table is desired is activated by the light pen, the octal number representing the truth table will be recorded on a punched card. This information may then be fed into another program which will deter-

mine the classical equivalence class descriptor of that function. The function routine may also be used (Fig. 14). The other approach mentioned earlier will enable us to obtain a descriptor which is not limited to six variable functions. This unlimited variable descriptor will be incorporated into the system represented in Fig. 3.

In order to give the designer some idea of how this partitioning is progressing, he may touch the instruc-

tion point labeled "UNPARTITIONED LOGIC." This restores on the screen all the logic which has not been partitioned, minus those positions which have been partitioned out, as shown in Fig. 15. If for some reason the designer needs to refer to the overall cluster of logic, he may energize the instruction touchpoint labeled "REDRAW LOGIC." This will draw all the logic blocks back on the screen with the partitioned section showing in its original posi-

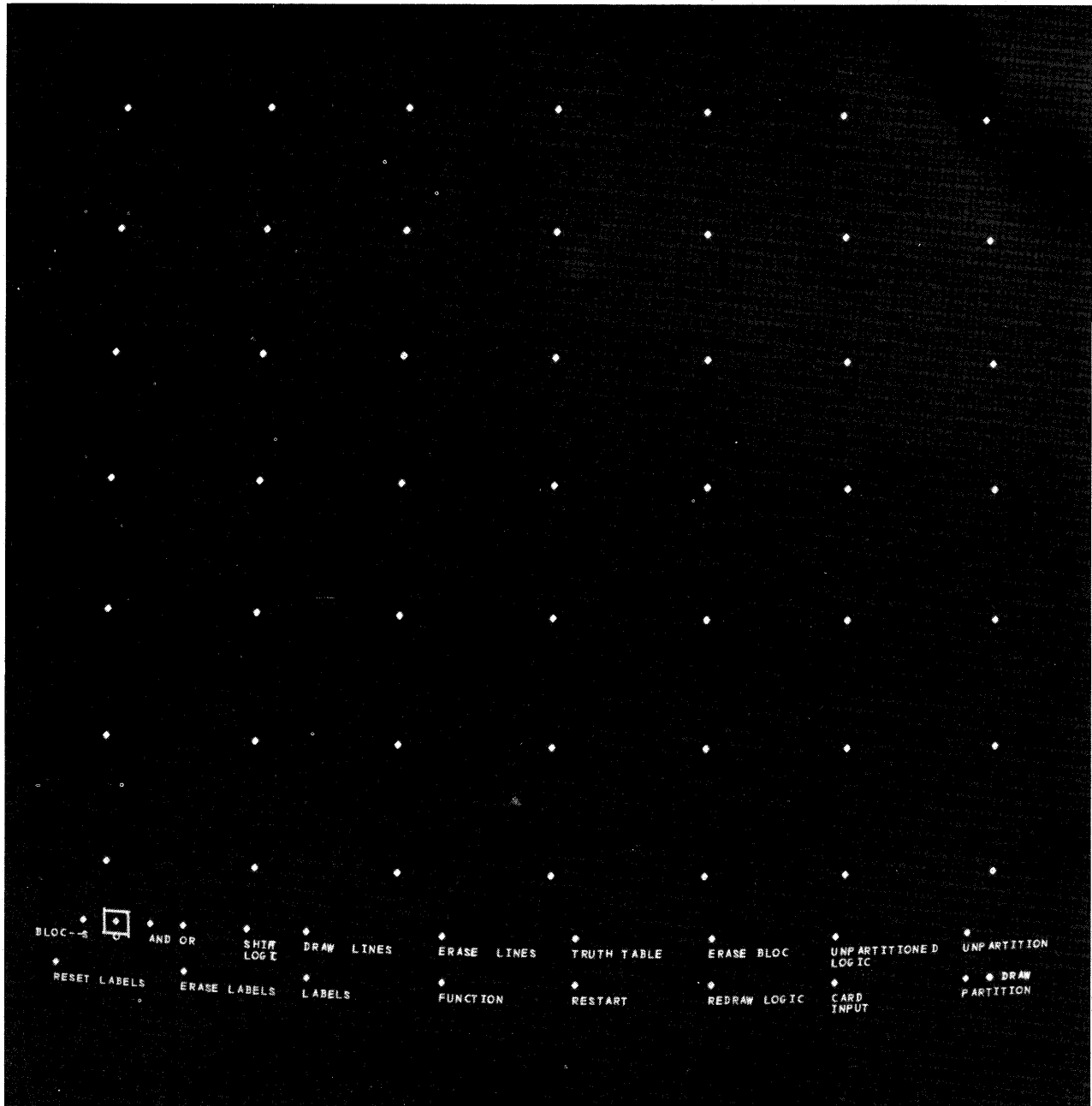


Figure 5. Light-pen instructions (*bottom*) and touch points.

tion but disconnected from the rest of the logic as indicated in Fig. 16. If the partition does not meet some arbitrarily established criteria, the designer may energize the instruction touchpoint labeled "UNPARTITION." He may then point his light-pen at

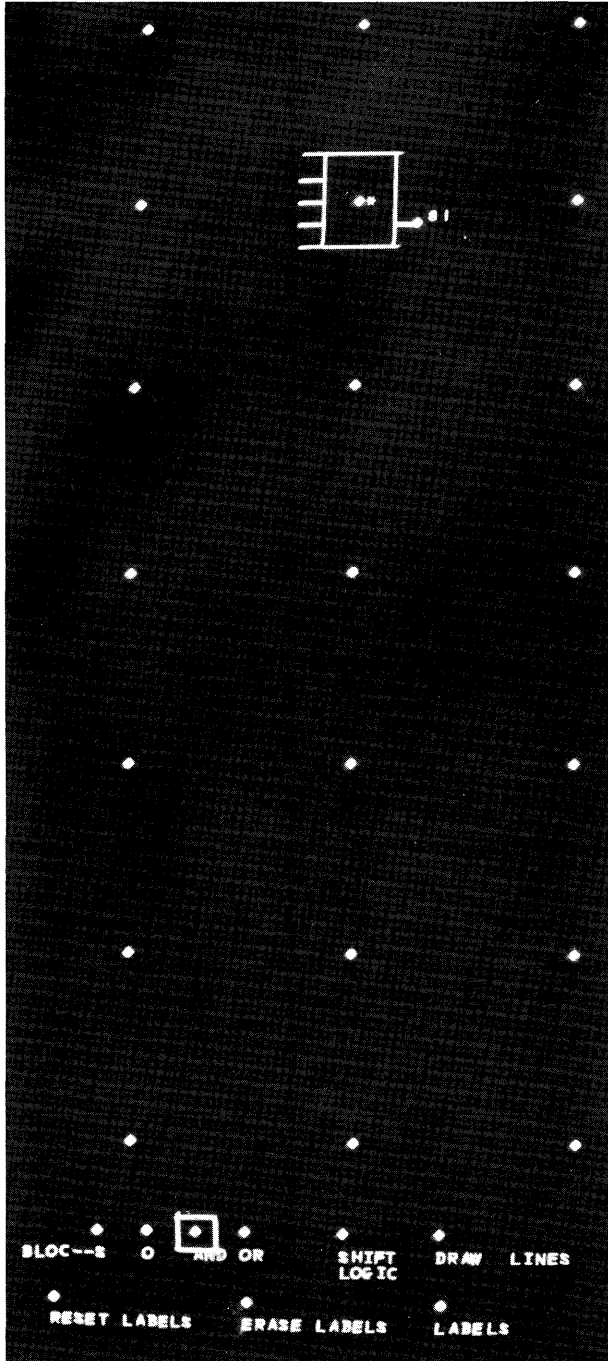


Figure 6. Logic block generated by firing light-pen at touch point.

each of the blocks that have been partitioned out, energize the instruction touchpoint labeled "RE-DRAW LOGIC," and the logic will then appear on the screen with the section that had been partitioned, reconnected as it was originally displayed. The designer then continues to partition the logic, calling out the instruction, "UNPARTITIONED LOGIC" until all the logic on the screen has been partitioned (Fig. 17).

The designer may operate on larger sections of logic by means of the touchpoint labeled "SHIFT LOGIC." The CRT then acts as a window through which the designer may look at any section of 49 blocks or less of the entire logic stored on the disk. Once "SHIFT LOGIC" is activated, another set of instructions appear at the bottom of the screen (see Fig. 18) which enables the designer to shift the logic on the screen, to the right or the left the number of blocks specified. Figure 18 shows the shifting instructions on the bottom of the CRT coupled with the logic as yet unpartitioned. Figure 19 shows this logic shifted three columns to the right. The logic for a complete computer can be visualized as being stored on a "scroll" which is seven logic blocks high by a couple thousand logic blocks wide.

All inputs to a 49-block cluster are either labeled as a primary input with letters or with the designation number of the logic blocks that feed the cluster (i.e., logic blocks on the part of the "scroll" not visible on the screen). The designer may then go through the partitioning technique for the cluster which is being viewed. After partitioning a 49-block cluster, there will be individual logic blocks around the edges of the screen (Fig. 20). These were not partitioned out but are connected to logic appearing other places upon the logic "scroll." He may then shift the logic and include these stragglers in partitions with logic blocks to which they are connected (Fig. 21).

Using the above technique, several hundred partitions may be taken out of the overall machine logic, their function described by the appropriate descriptor and the total number of function types thus greatly reduced. This process may be repeated by the same designer or by other designers in order to obtain an optimum set of logical elements to be used in the implementation of the computer being designed.

An important advantage of this approach from the designer's point of view is that it enables the indi-

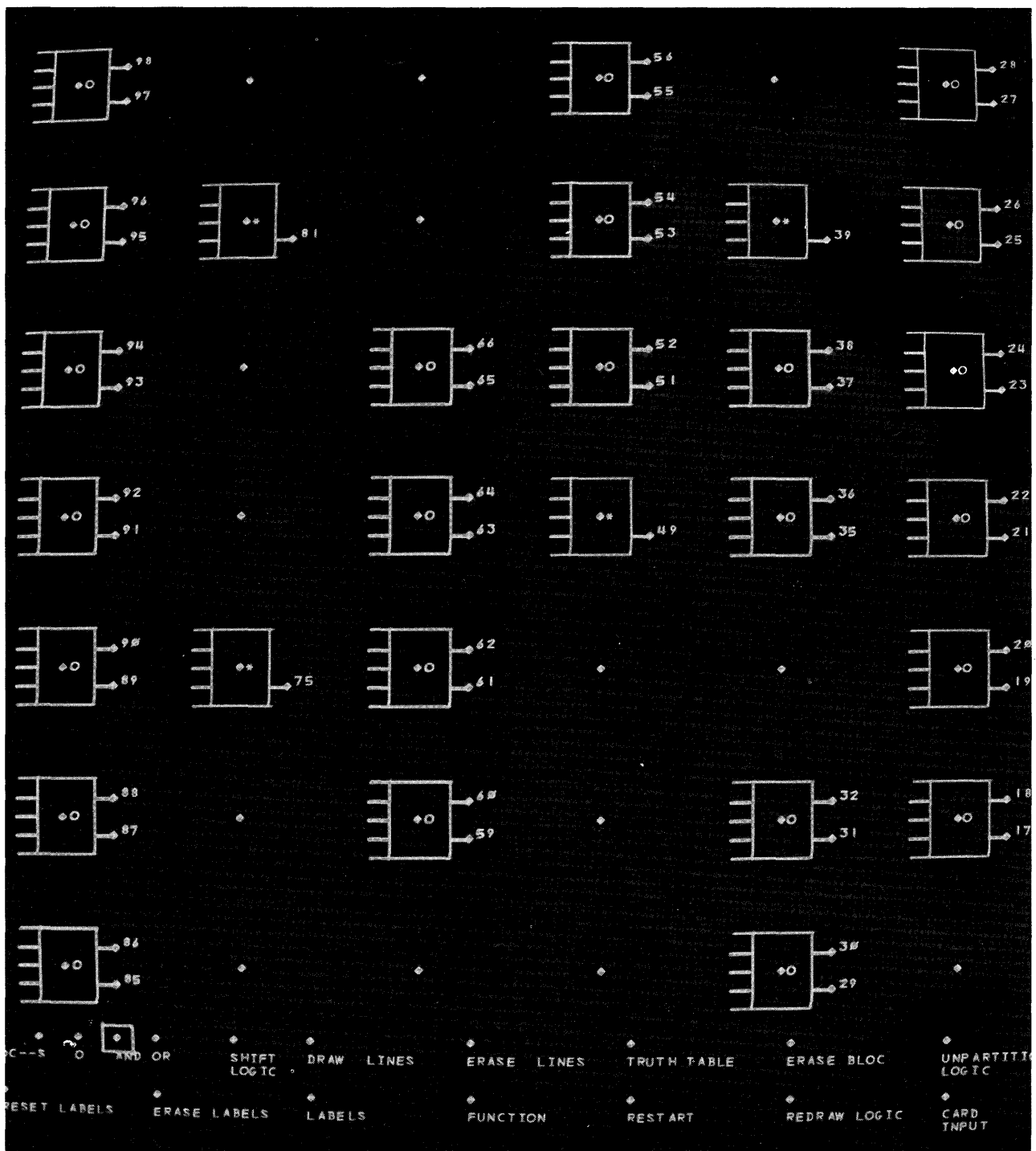


Figure 7. Complete logic block array generated by light pen.

vidual engineer to design as efficiently as possible. He is not restricted by package limitations or cell function constraints in his "raw" logic design. The functions partitioned out by one or more designers

may be evaluated, compared and combined into efficient multipurpose cells or Universal Logic Blocks. The cells may then be replaced into the logic, where the original partitions were removed, and connected as determined by the descriptor program which is now under development.

APPENDIX

LIBRARY SUBROUTINES

The library subroutines used to change the contents of the buffer, thus changing the pattern on the CRT, are CHAR and VECT.

CHAR(NX, NY, N, NALPHA, J, L)

NX, NY	X, Y coordinates of first character of message
N	number of characters to be displayed
NALPHA	name of the array which contains the message
J	an index which is incremented by the amount of buffer positions used to store message
L	buffer position where message is started

VECT (NX, NY, NXE, NYE, J, L)

NX, NY	coordinates of start of vector
NXE, NYE	coordinates of end of vector
J, L	as explained for CHAR

The other library subroutines used are:

WAIT	Causes machine to stop processing until there is a light-pen interrupt.
ACT	Turns off the mask on the 1620 so that information from the light-pen can enter the processor.
INACT	Turns the mask on so information from the light-pen won't interfere with the processor.
CORDNT	Determines the x and y coordinates of the character the light-pen is firing on.
CONR 2	This subroutine is used by CORDNT to transfer one word of information from the display memory to the

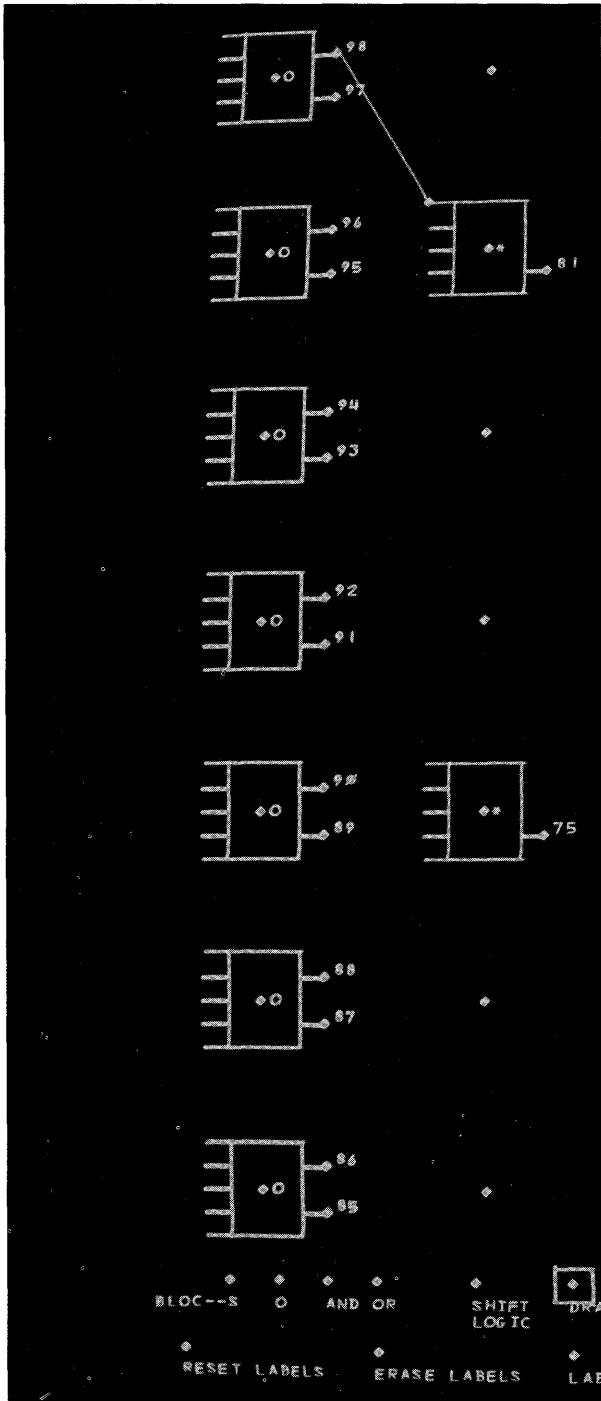


Figure 8. Two logic blocks connected by a light-pen using "DRAW LINES" routine.

processor memory. The word that is transferred depends on the character that the light-pen is firing at.

DTOC

Is a decimal to octal conversion subroutine.

FEINT

This is another subroutine used by CORDNT. It provides the processor with the display memory address and other information (such as which character is a narrative word) when there is a light-pen interrupt.

RESUME

Returns processor control to IR 1. When there is a light-pen interrupt control is transferred to IR 3. When the processor is under con-

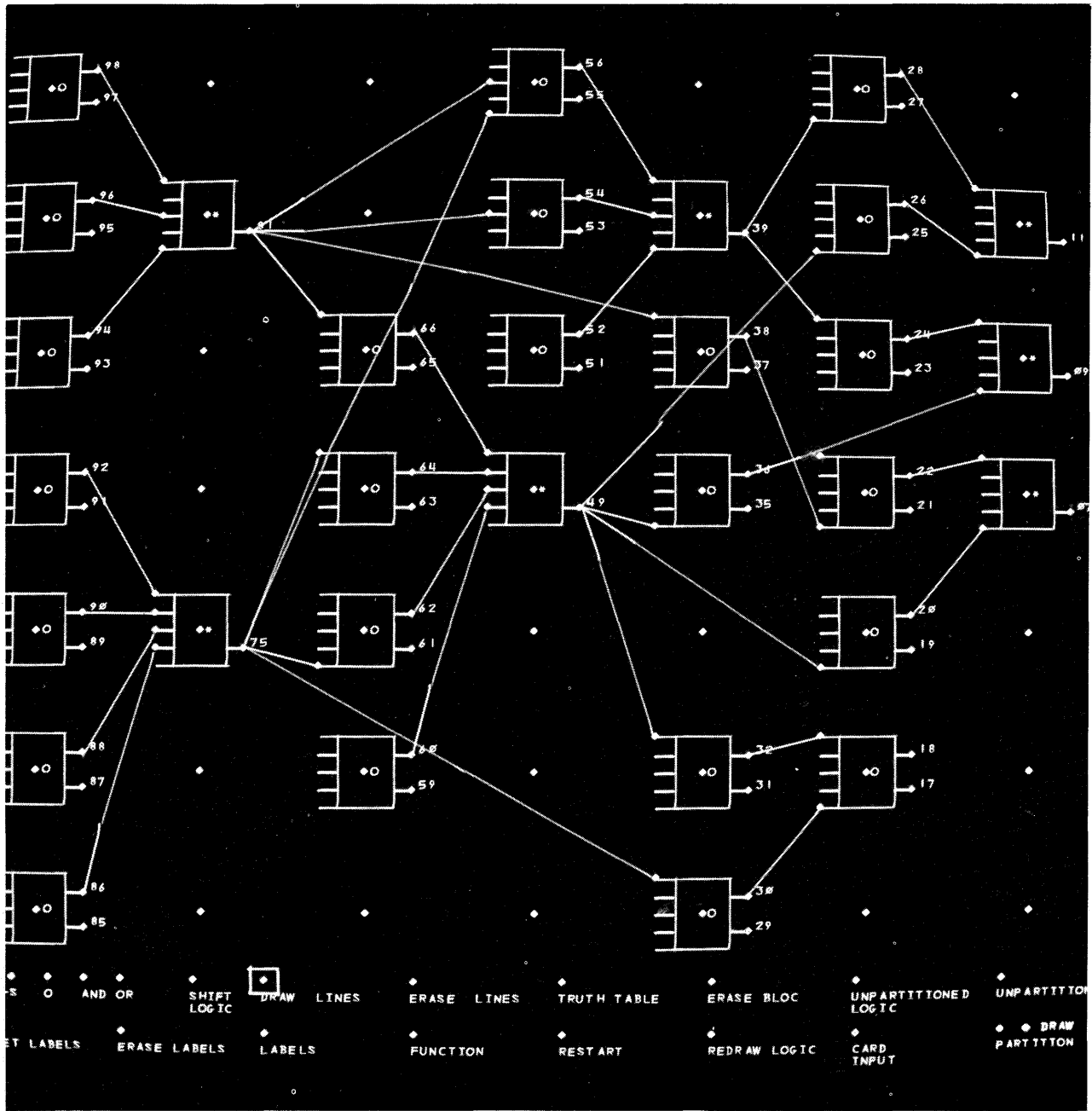


Figure 9. Complete logical array interconnected.

trol of IR 3 it cannot accept another light-pen interrupt.

QUINK Permits programs stored on the disc in core image to be loaded into storage much more quickly than the conventional FORTRAN 'LINK' statement.

DISCR Reads information from disk.

DISCW Writes information on disk.

ACKNOWLEDGMENTS

The authors wish to thank Messrs. T. Kameda and Y. N. Patt for their assistance during the summer of 1965 in obtaining an understanding of the classification of Boolean functions.

REFERENCES

1. D. Slepian, "On the Number of Symmetry Types of Boolean Functions of n Variables,"

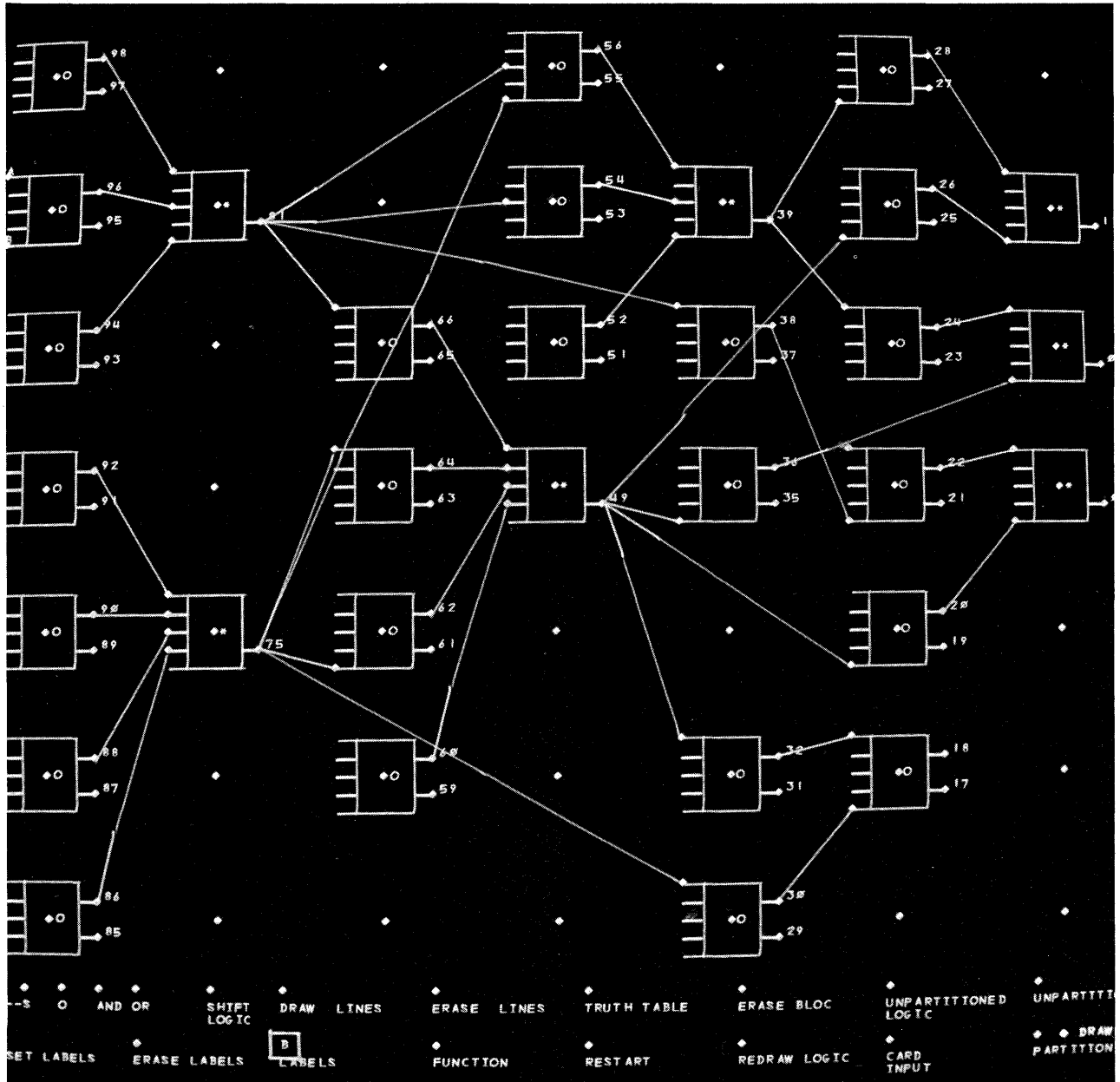


Figure 10. One logic block labeled using "LABELS" routine.

Canadian Journal of Mathematics, vol. 5, pp. 185-93, 1953.

2. S. W. Golomb, "On the Classification of Boolean Functions," *IRE Transactions on Circuit Theory*, vol. 6 (Special Supplement), pp. 176-86 (May 1959).

3. R. C. Minnick, "Cutpoint Cellular Logic," *IEEE Transactions on Electronic Computers*, Dec. 1964.

4. W. F. King III, and A. Guisti, "Can Logic Arrays be Kept Flexible?" AFCRL Report No. 65-547 (Aug. 1965).

5. I. E. Sutherland, "Sketchpad; A Man-Machine Graphical Communications System," MIT Technical Report No. 296 (Jan. 30, 1963).

6. D. T. Ross, "Implications of Computer-Aided Design for Numerically Controlled Production," MIT Report ESL-TM-212 (Sept. 1964).

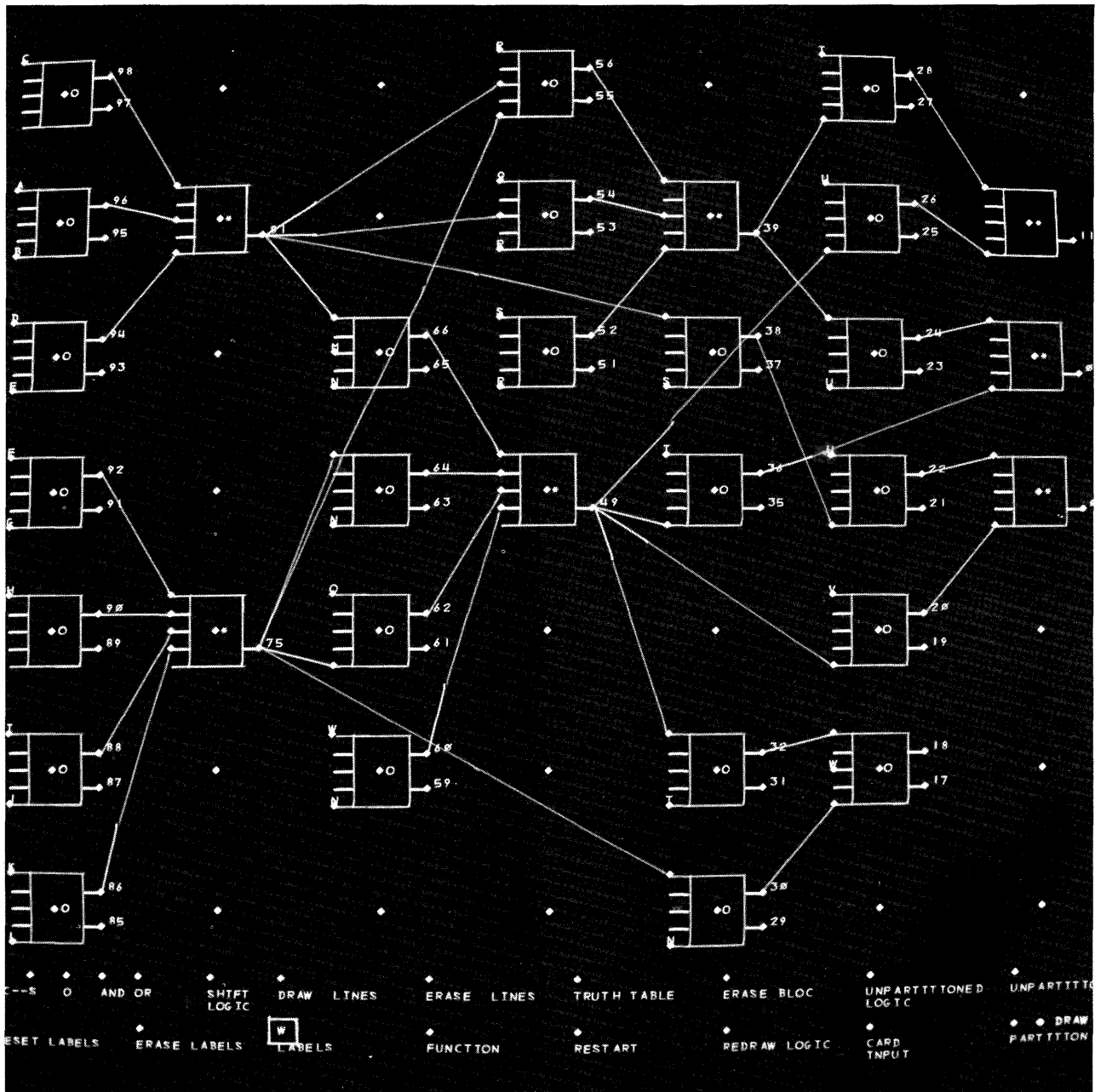


Figure 11. Completed logical array labeled.

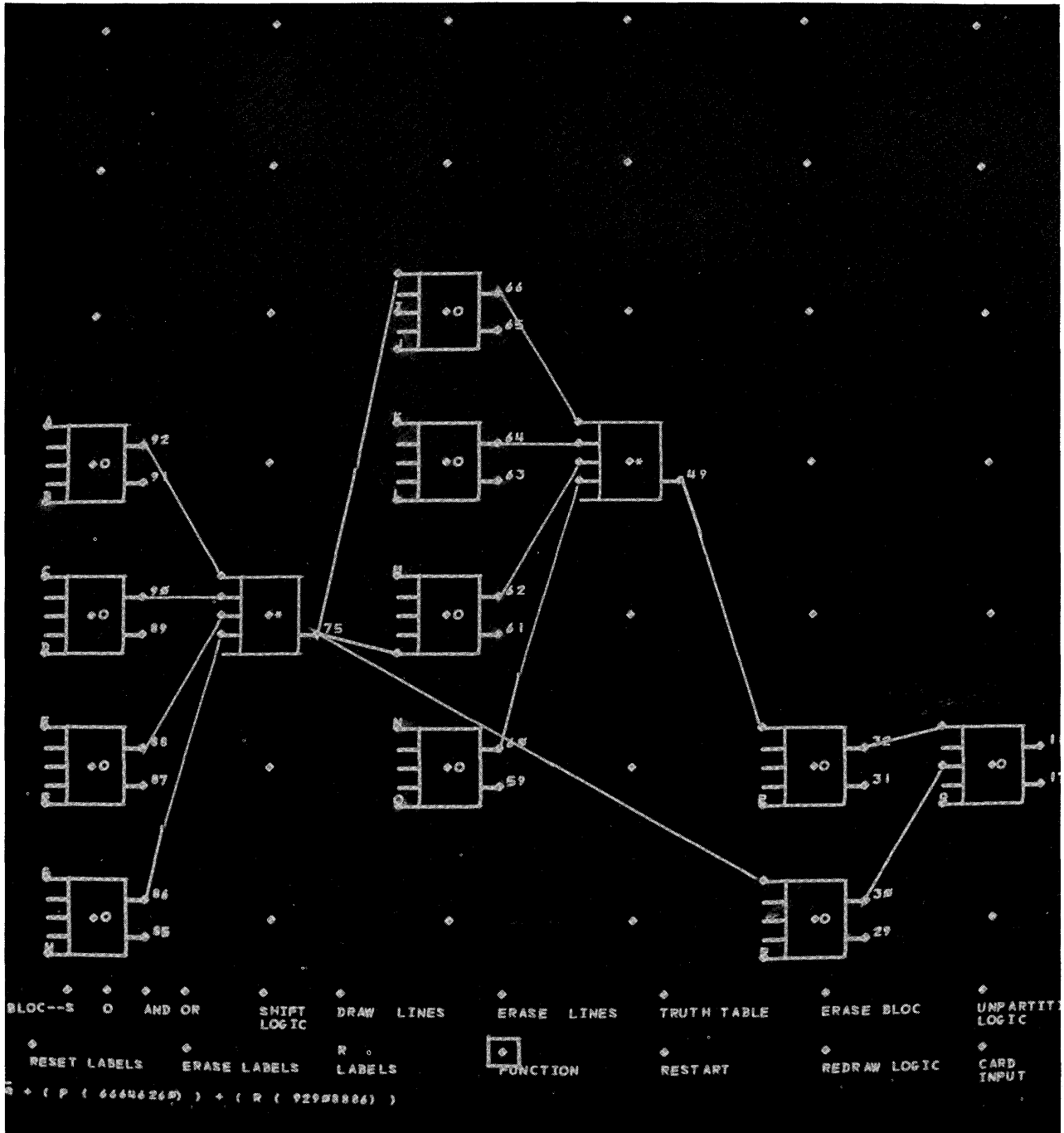


Figure 12. Example of a three-level function from unpartitioned logic.

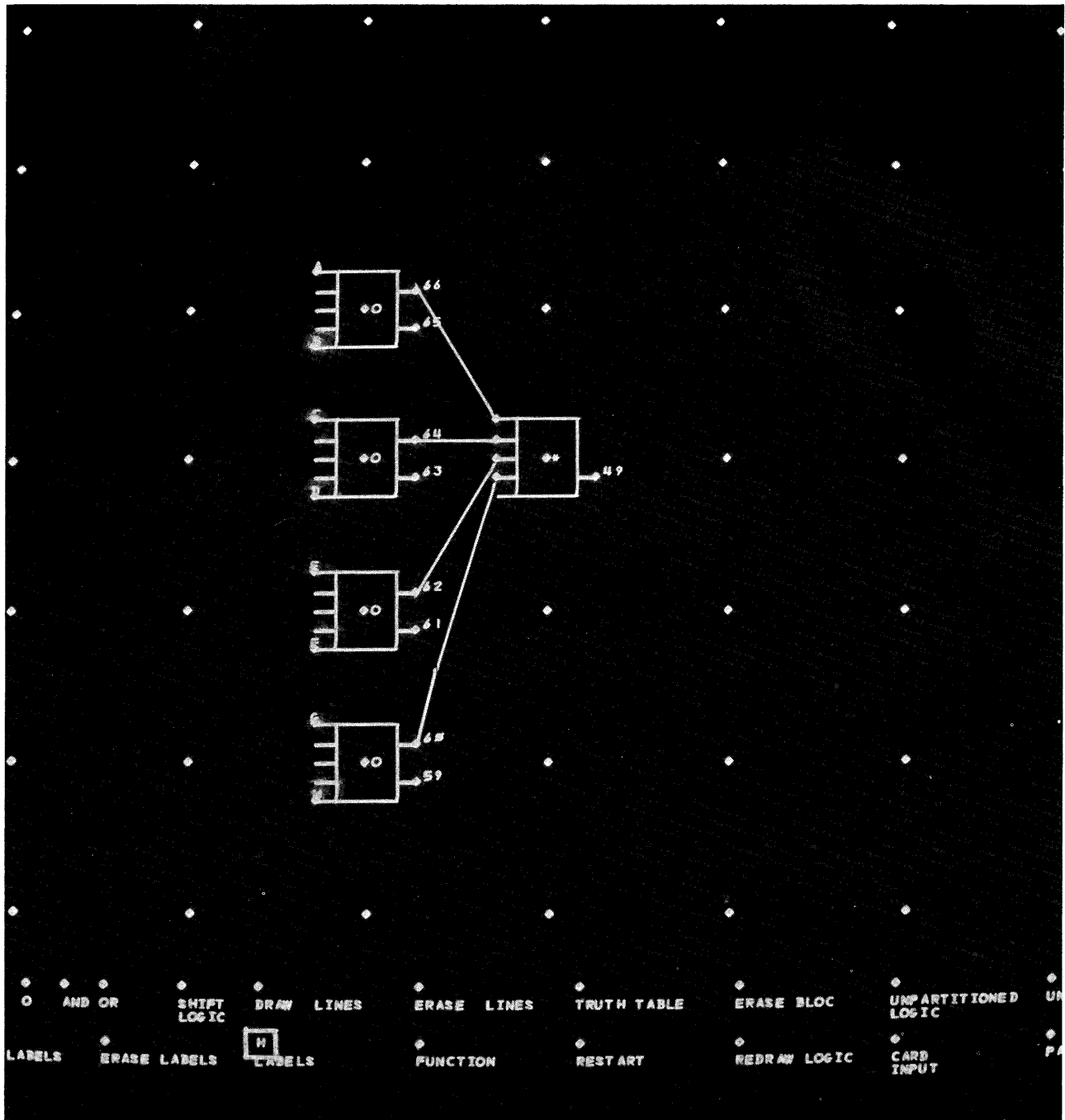


Figure 13. Partitioned-out cluster of logic.

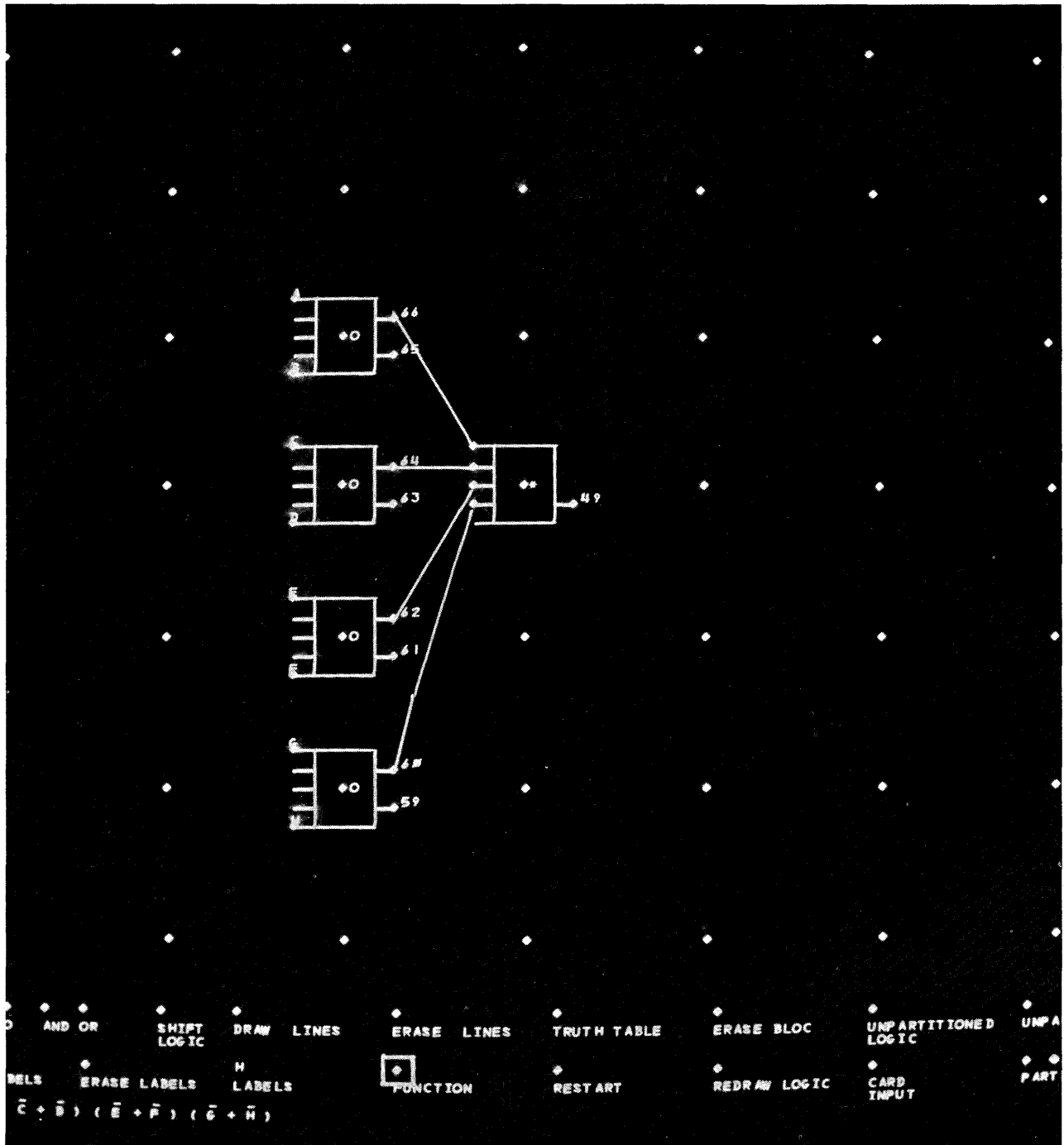


Figure 14. Partitioned-out cluster of logic with output function.

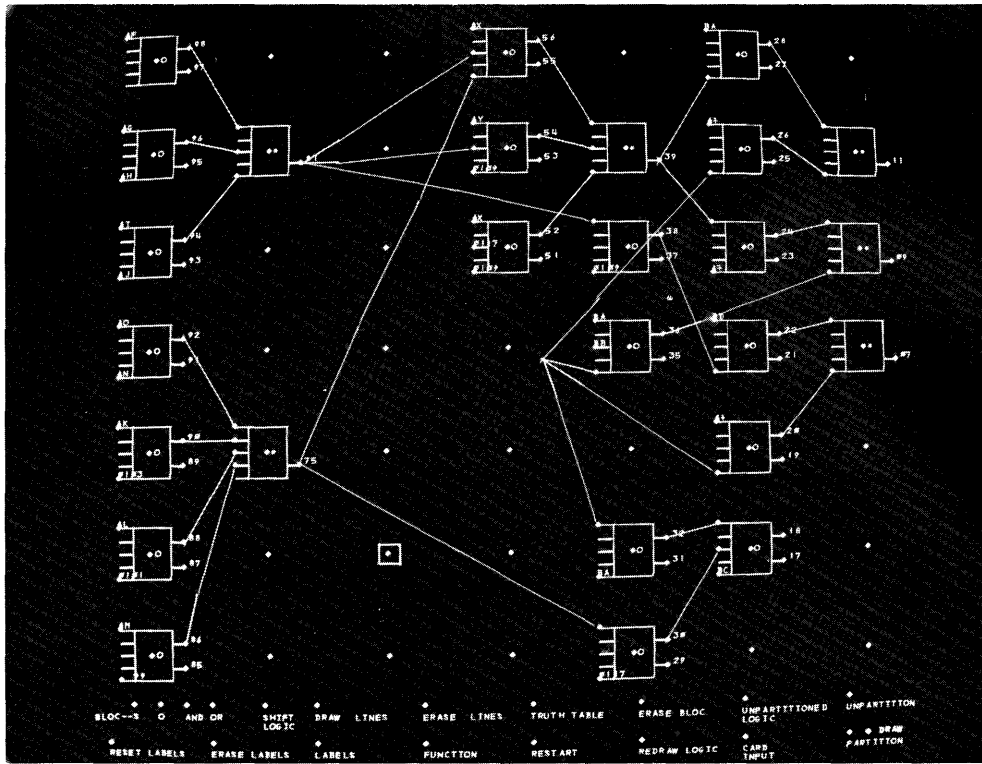


Figure 15. Unpartitioned logic with one partitioned cluster missing.

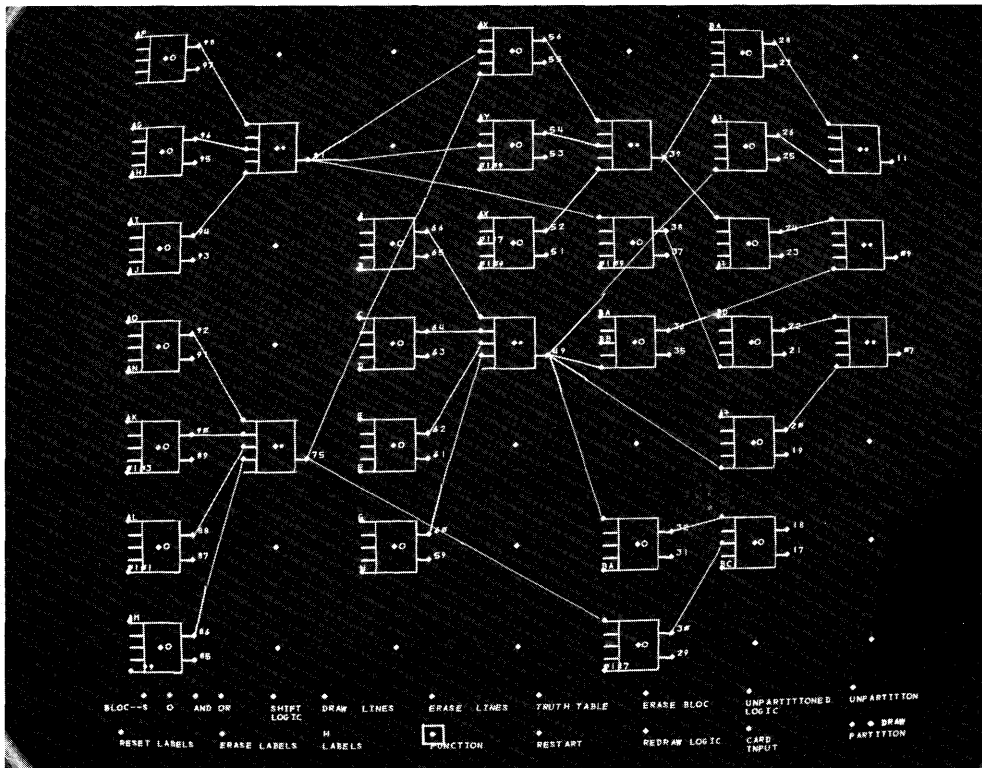


Figure 16. Logical array with partitioned cluster disconnected.

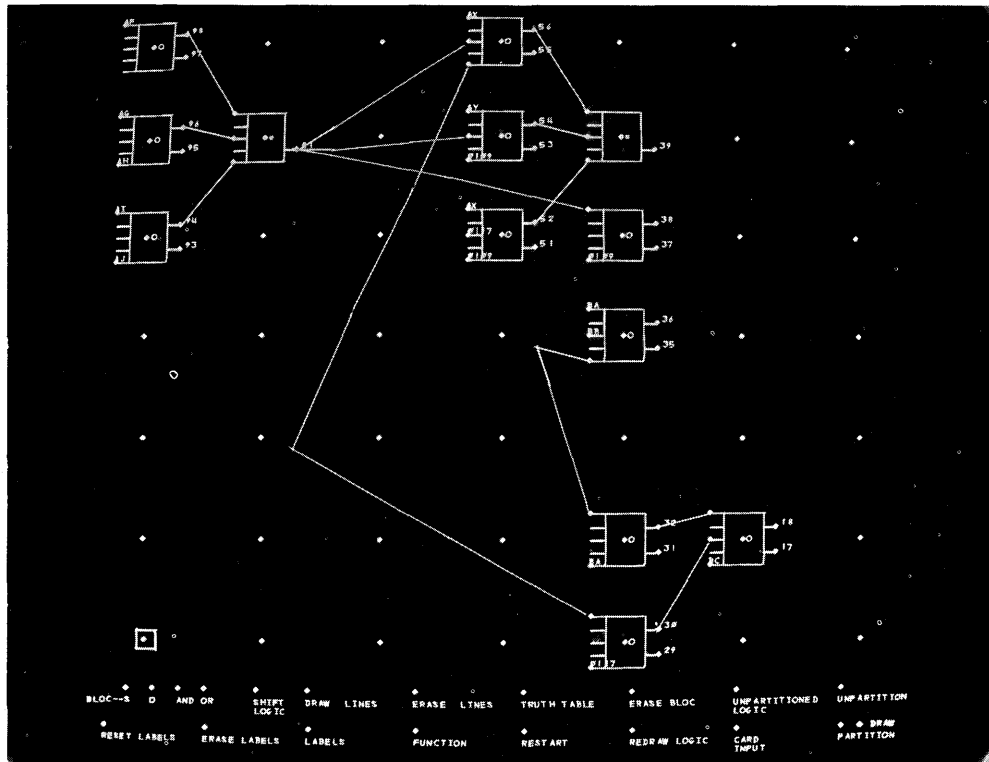


Figure 17. Unpartitioned logic with several partitioned clusters missing.

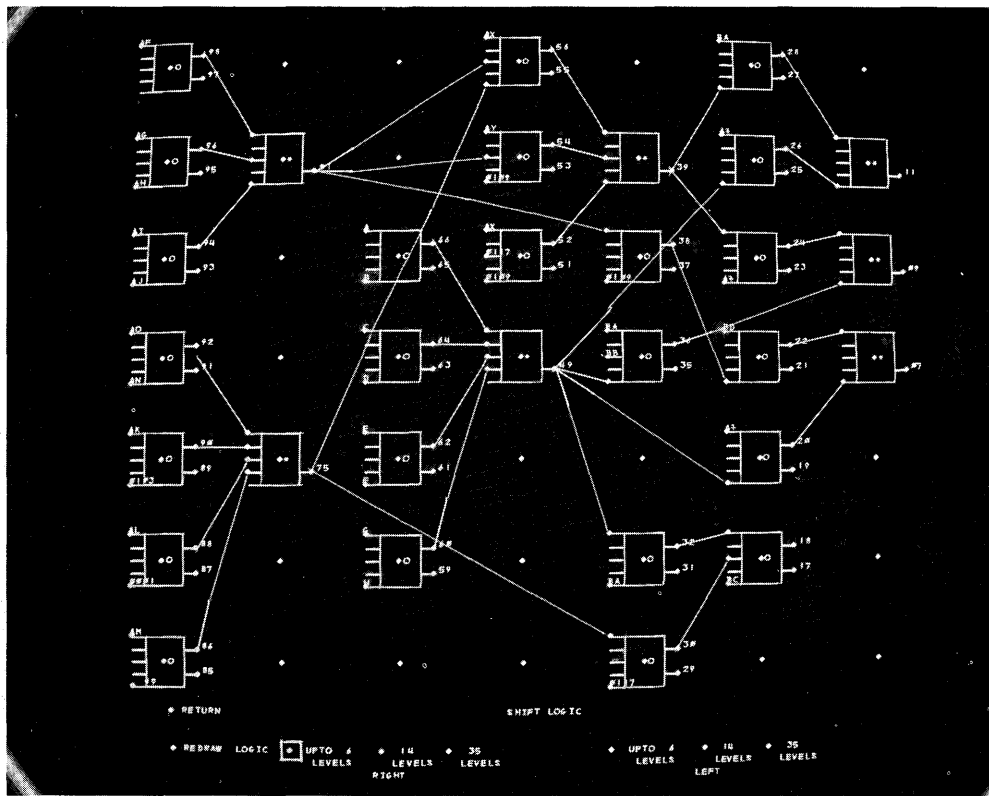


Figure 18. Logical array shown with "SHIFT" instructions shown at bottom of CRT.

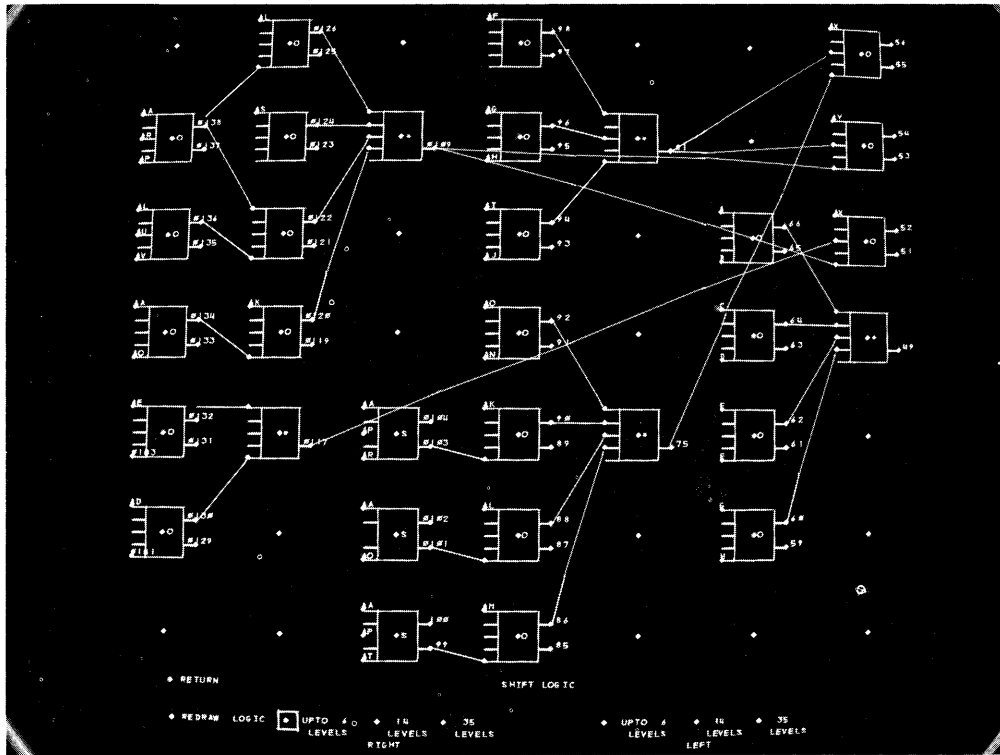


Figure 19. Logical array shifted three columns to the right.

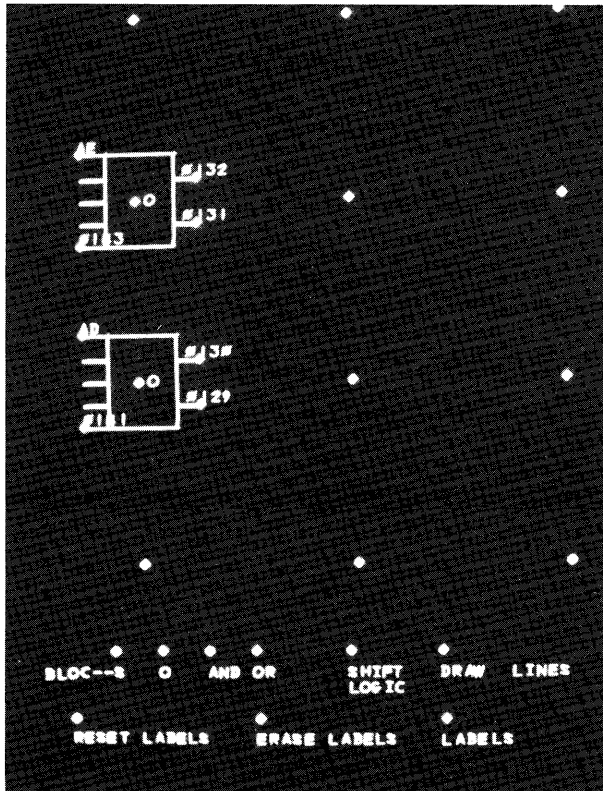


Figure 20. Logic blocks remaining on screen after partitioning.

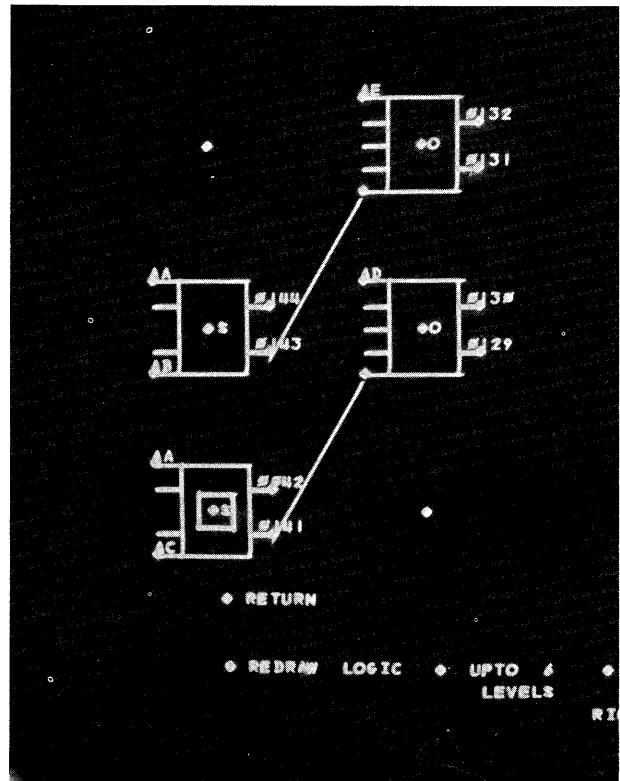


Figure 21. Logic shifted to the right of screen.

A COST/PERFORMANCE ANALYSIS OF INTEGRATED-CIRCUIT CORE MEMORIES

Dana W. Moore

*Honeywell Computer Control Division
Framingham, Massachusetts*

INTRODUCTION

The purpose of this paper is to give added insight into the influence of memory organization on system cost and performance, to show how various configurations may be best suited to satisfying a particular set of requirements, and to indicate the effects of low-cost integrated circuitry on the storage array/circuit cost tradeoffs available with varying organizations. To accomplish these objectives, five typical organizations were selected, and each was dealt with in sufficient detail to allow subsequent predictions of cost and performance. Those chosen are applicable to random-access ferrite-core memories, and each design makes use of integrated circuitry to the extent permitted by the devices which presently may be procured in volume quantities. The configurations discussed are as follows:

- 3D, four-wire (3D, 4W)
- 3D, three-wire (3D, 3W)
- 2½D, three-wire (2½D, 3W)
- 2½D, two-wire (2½D, 2W)
- 2D, two-wire (2D, 2W)

Cost estimates for systems incorporating these organizations were obtained by examining each individually. First, however, circuit modules were designed to perform the basic high-current switching, decoding, and sensing functions required in core

memories. These modules, or slightly modified versions, were then assembled to create a family of systems about each organizational structure. Costs for the modules were estimated on the basis of typical volume procurement and assembly costs currently prevalent in the industry. Total cost is then tabulated as a function of memory capacity for comparison with alternative approaches. Totals include all decoding selection and sensing circuitry and estimated cost of third and fourth wires which may be included in the storage array, depending upon the organization considered. Not included are costs of the cores and the basic two-wire array necessary for all configurations considered, regardless of organization. Also excluded are costs of registers, timing and control logic, hardware, power supplies and system assembly, since their contribution is relatively small and the functions performed are similar in each design.

The result is a cost comparison of functionally equivalent memory systems representing each of several organizations. The effect on this comparison of changing memory capacity is examined in detail. All designs except the 2½D, two-wire mass memory operate at speeds of the order of 1 μ sec.

The following section describes the circuit modules common to all designs, their use and estimated unit cost. Subsequent portions show how the modules are most effectively used or modified to realize each of the five configurations, and concluding remarks give

a comparative summary of all results as well as general effects of changing speed requirements.

CIRCUIT FUNCTIONS

Each design requires arrays of matrixed switching circuitry to route large drive currents into selected lines of the storage arrays. In the past, large memories of this type were most economically constructed by using a minimum quantity of relatively high-performance drive circuitry. Large stacks were employed containing 5000 or more cores per drive line and, consequently, the lines were necessarily terminated in their characteristic impedance. Four diodes per line were required to isolate the termination and to permit use of identical current sources for both read and write currents. Typical line impedances required memory voltages in excess of 60 volts to provide 400 ma current flow and, although selection-circuit component costs were high, these voltages and currents could be handled by discrete circuitry. This, coupled with high assembly costs and low packing densities of discrete modules, made the approach using a minimum of circuit functions the most attractive.

The advent of integrated circuits, however, has economically obsoleted systems designed about this approach. Monolithic technology has significantly lowered the cost of low-voltage functions, reduced assembly costs, and increased packing densities. Consequently, an overall gain may be realized by employing a larger number of cheaper circuit functions to reduce the power requirements placed on each. Drive lines may be shortened to the point where termination is unnecessary. Supply voltages then may be reduced to some value which yields adequate current stabilization through a series limiting resistor or a lower value determined by rise-time considerations when active current sources are employed.

Selection Switch

Drive matrices in all designs except the large $2\frac{1}{2}D$, two-wire system incorporate the above philosophy, and these matrices are assembled with circuit modules containing four read-write pairs on a $2\frac{1}{2}$ " square card. The switches are transformer-coupled to logic circuitry and capable of handling 400 ma currents at 30 volts. In addition, sufficient integrated gating and amplification is provided on each module to permit selection of 1 out of 16 pairs without external gating. It is estimated that the

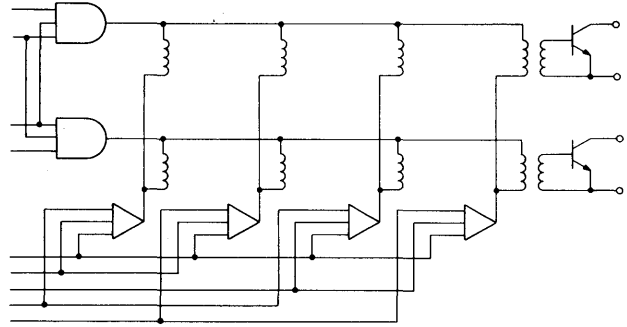


Figure 1. Selection switch module.

switch module could be manufactured in quantity at a cost of \$40. A schematic of the circuit is shown in Fig. 1.

Diode Matrix

The switches are used with passive current sources in a two-diode-per-line matrix where the diodes are purchased in standard integrated-circuit packages for approximately 20¢ each. A schematic of the matrix used is shown in Fig. 2. The maximum size of this configuration for 1- μ sec operation is 256 unterminated lines of 1024 cores each. Discarding the termination permits use of a low-voltage supply (24 volts) thereby reducing component costs within the matrix. The constraint on matrix size results from the increasing bus capacity, which would exceed 1000 pf for 16 lines of this length in a four-wire array.

Sense Amplifier

Lower voltage excursions within arrays help reduce sense-line perturbations to the point where inte-

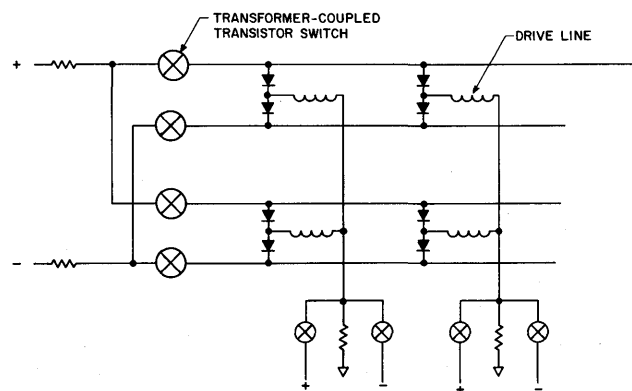


Figure 2. Unterminated drive matrix with passive current source.

grated sense amplifiers are practical. This is particularly important in 2½D configurations where nonsquare arrays increase coupling to drive lines in the digit matrices. Sense amplifiers compatible with these designs may be purchased in quantity at \$20 per unit and are suited for cycle rates in excess of 1 mHz. They may service a maximum of 4096 cores, corresponding to a normalized cost of ½¢ per core in systems of 4096 addresses or greater. In 3D systems, however, care must be taken to limit current rise-times when low-voltage sense amplifiers are employed. Otherwise, large difference signals may appear at amplifier terminals for certain data patterns, and high-speed inhibit transients might then cause permanent damage to the sensing circuitry.

Current Switch

For 3D systems, a current switch is required to handle data-modulated inhibit currents. The circuit used to provide this function employs components similar to those in the selection switch and includes a discrete transformer-coupled transistor switch driven by an integrated-circuit power gate. It is estimated that these components may be procured, assembled, and tested for \$6.25 per switch.

In summary, a number of circuit modules were specified, designed, and cost-estimated. The functions which they perform represent the major expense in most large magnetic-memory systems, and they may be organized or modified in a variety of ways to build families of systems of varying capacity and organization. The functions are listed in Table 1 along with their costs and the organizations in which they are used. In cases where functions are modified to suit unique requirements of particular configurations, those changes are described in subsequent sections which discuss each design in some detail.

3D, Four-Wire Configuration

The 3D configuration of selection circuitry probably is, and most certainly has been, the most

popular method of accessing coincident-current core arrays.¹ Although the organization requires fewer circuit functions than any of those discussed, it is dependent upon a more complex stack. Drive lines of each axis thread through all bits of the data word, and individual inhibit and sense wires thread through all cores associated with a bit position in the data word. Currents in inhibit wires then are modulated by incoming data or data about to be regenerated to cancel "write one" drive currents.

The design consists of the two drive matrices, each composed of selection switches and diodes, described previously. Lines are unterminated and limited in length to 1024 cores, and this results in a family of maximum module capacities where word length equals the number of cores per line divided by the square root of the number of addresses. In this case

$$B \text{ max} = \frac{2^{10}}{\sqrt{A}}$$

Inhibit drive circuitry uses one current switch per 4096 cores and an external resistor. Cycle times of the order of 1 μsec could, then, be achieved with these circuits, a 24-volt supply (not included in costs) and 30-mil core arrays.

Shown in Fig. 3 are curves which indicate the effects of changing capacity on the costs of systems organized in this fashion. The costs at each point represent an optimum assemblage of the functions described above, and the relative contributions of selection, regeneration, and array costs are shown for each capacity.

Also shown are the maximum number of bits per module for each capacity where a module refers to a single stack assembly. In those units requiring matrices of unequal size (i.e., 2048, 8192, etc.), two matrices are employed on one axis such that all lines may contain the maximum number of cores. Maximum word length for these modules is, consequently, equal to that of the next smaller capacity.

As would be expected, selection circuit costs be-

Table 1. Memory-Circuit Functions

Function	No. Functions per Module	Cost per Function	Used in:				
			3D, 4W	3D, 3W	2½D, 3W	2½D, 2W	2D, 2W
Selection switch	8/2½" card	\$ 5.00	Yes	Yes	Yes	Mod	Mod
Diode matrix	16/flat pac	.20	Yes	Yes	Yes	Mod	Mod
Sense amplifier	1/flat pac	20.00	Yes	Yes	Yes	Mod	Yes
Current switch	6/2½" card	6.25	Yes	Mod	No	No	Mod
Inhibit wire	1/core	.005	Yes	No	No	No	No
Sense line	1/core	.005	Yes	Yes	Yes	No	No

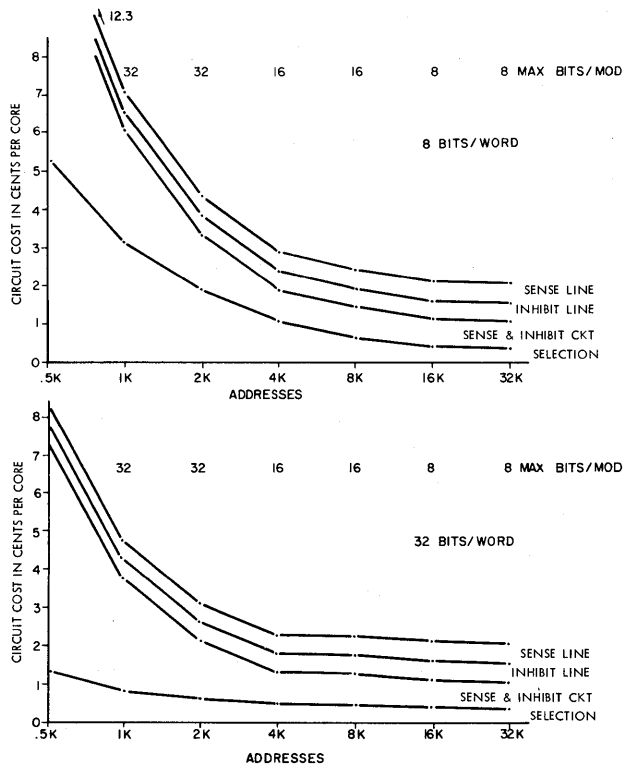


Figure 3. Circuits costs for optimum 3D, four-wire configurations.

come less significant as capacity increases, and this is made more apparent when considering the additional array and assembly costs required to complete a module. These might typically amount to 1¢ for a 30-mil core and 0.5¢ to provide control logic, registers, hardware and assembly for a 250,000-bit unit.

3D, Three-Wire Configuration

This configuration is identical to the 3D, four-wire approach except for the fact that a single wire is routed through the core array in such a way that it may be time-shared to provide both sense and inhibit functions. Although regeneration-circuit costs increase with the addition of components used to combine the two functions, overall savings can result from the lower three-wire stack cost. A unique array is required which, in spite of its dissimilarity to 2½D and other 3D stacks, imposes no design problems other than a small initial development cost to establish a source. This stack, once designed, would require fewer connections to associated driving and sensing circuitry than any of the others discussed.

In the scheme proposed for comparison, a family

of systems are considered using selection circuitry identical to that in the 3D, four-wire case, for stack wiring and sense and inhibit circuitry. The same restrictions regarding line length and matrix size apply. Line capacity, however, is lower here than in the four-wire situation and, consequently, line length might be extended to take advantage of the shorter propagation time.

A circuit module was designed to provide sense amplification and inhibit currents for two bits of 4096 addresses each (see Fig. 4). Both source and sink ends of the inhibit circuit are placed on the same module to keep the high-current loop area small. A full select current is supplied initially at the source end which then divides in two parallel branches of the sense-inhibit wire. Current division is assured by a balance transformer placed at the opposite, or sensing, end of the line. Series diodes are placed at the terminals of the balance transformer to isolate it from the sensing path during readout and to provide a high impedance for rapid transformer recovery. A sense amplifier is then directly coupled to the line at the balance-transformer end and, in this configuration, the signals to be sensed are essentially identical to those which would occur with a separate sense winding.

The module which was selected to provide the above function is composed of devices used in the

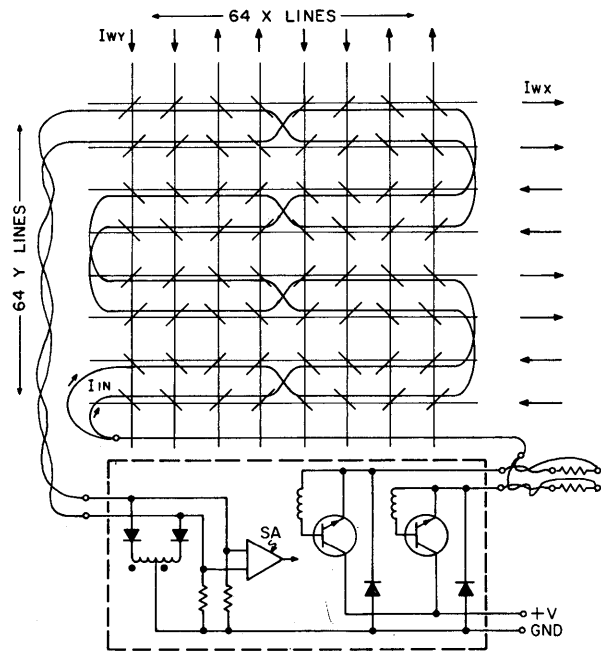


Figure 4. Sense and inhibit circuitry for two bits of 4096 addresses each.

selection switch, diodes, and the integrated sense amplifier. Additional diodes are included to discharge the line inductance through the current-limiting resistor, and two transformer-coupled switches are connected in parallel to provide the full select inhibit current. Cost of the configuration was estimated at \$41 per data bit for each 4096 addresses.

A circuit cost increase of 0.2¢ results over the four-wire approach, but an overall saving of 0.3¢ is realized when the 0.5¢ cost of the discarded wire is considered.

Figure 5 shows plots of memory cost vs capacity for this design in fashion similar to that of Fig. 3. Cycle times for these configurations also are of the order of 1 μsec when used with 30-mil core arrays.

2½D, Three-Wire Configuration

It is commonly known that the high cost of adding a fourth wire to high-density core arrays in many cases justifies a 2½D organization.² The descriptor 2½D refers to the method of coincident element selection, and identifies two- or three-wire memories in which inhibit lines and drivers are replaced by an independent selection matrix for each bit of the data word. In these instances, the cost of repeating the digit matrix for each bit is more than offset by

the consequent reduction of core array complexity from four wires per core to three. Furthermore, if care is taken to minimize delays of address fanout to the multiple-digit matrices, higher speeds may be obtained with a given storage element than in large 3D systems because it is unnecessary to cancel write currents with an overlapping inhibit current. Instead, in the 2½D scheme, individual digit matrices are modulated by incoming data. The scheme is made practical by the advent of low-cost integrated circuitry. In addition, discarding the inhibit function permits efficient use of a read-write interchange technique which reduces by half the number of drive wires to be selected on one axis of the stack.

Because of the repetition of the digit matrix, non-square core arrays are employed to optimize total selection-circuit costs. Consider a 2½D system where the X matrix services all cores and the Y matrix is repeated for each bit. Total selection circuit cost is, then,

$$T = 2K_s \sqrt{L_x} + DL_x + B(2K_s \sqrt{L_y} + DL_y)$$

- where K_s = cost of a read-write switch pair,
- D = cost of matrixing component,
- L_x = number of X lines,
- L_y = number of Y lines per bit
- W = numbers of words, and
- B = number of bits per word.

Since reentrant Y lines and read-write interchange logic are employed,

$$W = 2L_x L_y$$

Substituting this expression into the total cost function yields

$$T = 2K_s \left[\sqrt{\frac{W}{2L_y}} + B \sqrt{L_y} \right] + D \left[\frac{W}{2L_y} + BL_y \right]$$

The first term represents selection-switch costs, which are proportional to the square root of the number of lines selected. This term is minimized when

$$L_y = \frac{1}{B} \sqrt{\frac{W}{2}} \quad \text{or} \quad \frac{L_x}{L_y} = B^2$$

The second term accounts for costs which are proportional to the number of lines (such as diodes, or transformers if used), and its minimum occurs when

$$L_y = \sqrt{\frac{W}{2B}} \quad \text{or} \quad \frac{L_x}{L_y} = B$$

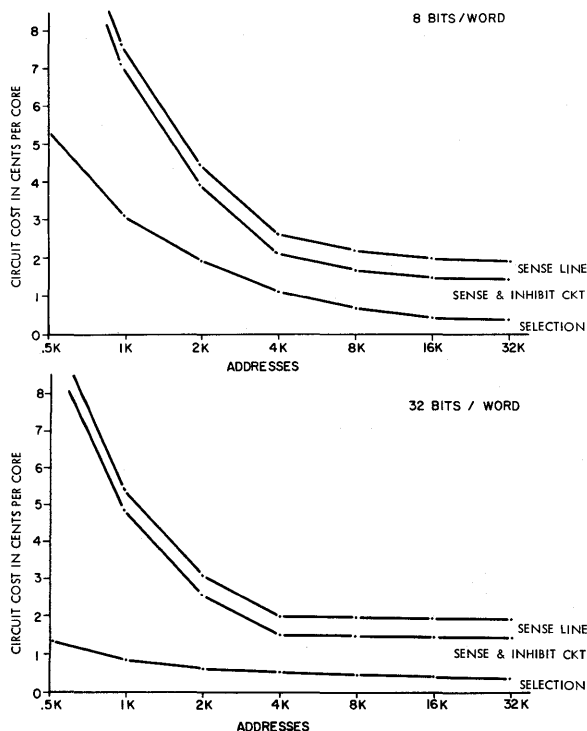


Figure 5. Circuit costs for 3D, three-wire configurations.

The optimum aspect ratio for the total expression is a function of the relative magnitudes of D and K_s but must fall between B and B^2 . For large systems where the selection switches are used more efficiently, or if D is made large by a requirement of a transformer per line, the latter term will dominate and the optimum ratio will approach B . For small systems, the converse occurs as selection switch costs become more significant, and the best ratio will approach B^2 . When a large ratio is required, L_x may exceed the maximum matrix size. In these instances, the ratio of $L_x = B^2 L_y$, as determined by selection switches, is invalid. Instead the expression

$$N^2 \frac{L_{x \max}}{L_y} = B^2$$

applies, where N is the optimum number of X matrices, each of which selects one of $L_{x \max}$ lines.

The total number of addresses then is

$$W = 2NL_{x \max} L_y$$

and

$$L_y = \left[\frac{W^2}{B^2 4L_{x \max}} \right]^{1/3}$$

This value for L_y then will minimize the first term of the total cost expression when the number of matrices required is a multiple of X .

Determining the best overall ratio is, of course, simplified by the fact that values for L_x and L_y are limited to those numbers which are powers of two, and possible solutions are so few that the ratio may be determined iteratively.

Other restrictions include practical considerations such as limitations on stack form factors; or solutions may result which would require an excessive number of circuit module types to generate a family of systems each of which is optimally organized for varying capacity. Also, a third level of matrixing often is employed in the selection of switch pairs, and this represents a cost term which is disproportionate to the number of switches as matrix size changes. Including this factor would require that the total cost expression include a third term proportional to the fourth root of the number of drive lines.

The points plotted in Fig. 6 represent circuit costs of $2\frac{1}{2}D$, three-wire systems built with the same 400 ma switching matrices as used in the 3D designs. Regeneration-circuit costs include the \$20 integrated sense amplifier and sense line for each 4096 cores.

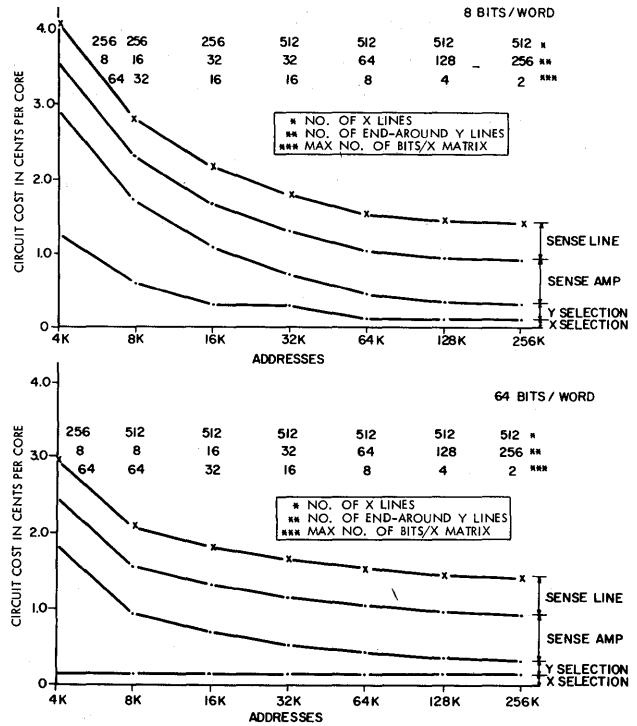


Figure 6. Circuit costs for optimum $2\frac{1}{2}D$, three-wire configurations.

Each point represents the most economical arrangement of these functions within the confines of electrical restrictions such as capacitive limitations on matrix size and reflections in excessively long, unterminated lines. Matrix size is consequently limited to 256 lines of 1024 cores each. End around (re-entrant) Y line length limits the number of X lines (L_x) to 512, and the allowable X line length determines the maximum word length per stack. All configurations operate at $1 \mu\text{sec}$, and the best aspect ratio for each capacity is shown by the number of lines in each matrix. The results are consistent with the foregoing discussion in that optimum ratios (where unrestricted) fall between B and B^2 . The ratios increase towards B^2 as the number of addresses decreases and increases again as word length increases.

$2\frac{1}{2}D$, Two-Wire Configuration

As shown in the previous section, large $2\frac{1}{2}D$, three-wire system cost approaches an asymptote composed principally of array and sensing costs. Consequently, when capacity requirements are large ($> 10^6$ bits), it is sometimes practical to institute a dollar-speed tradeoff to fill the void between rotating machinery and high-speed, random-access electronic

storage. Systems commonly described as mass stores exemplify such a tradeoff where $2\frac{1}{2}D$ selection is employed without a third sense wire.³ Sensing circuitry, although more complex, is integrated into the digit matrix in such manner that many more than 4096 cores may be monitored by a single amplifier. The added intricacy of the sensing function results from the requirement to discriminate a low-level core response from large signals generated by half-select drive currents flowing in the same wire as used for sensing.

Cycle times are generally in the 2- to 8- μ sec range for several reasons. One is the incentive to use a low-drive (but slower) core to reduce the cost and power of the digit-selection matrix which must handle full select currents. These currents divide and flow through two legs of a balanced drive/sense line configuration. Second, large sense signals result from digit current transients, and drive currents must necessarily be staggered. Third, a DC offset voltage appears across the sensing terminals for the duration of the read cycle. Consequently, added time is required to either differentiate or DC-restore the incoming signal. Fourth, drive lines are long, and current in the digit axis must stabilize before a core is switched by a current in the alternate axis.

A system of this organization was examined which appears to be a promising approach to the problem of isolating the sensitive sense circuitry from disturbances in the selection matrix. The design was carried out in sufficient detail to permit reasonably accurate estimates of component costs which, in turn, were tabulated in optimum configurations.

A low-drive core is used in the contemplated design such that the large matrix common to all bits need only switch 300 ma currents. Consequently, it is feasible to use the low-voltage selection switches in a matrix of terminated lines containing up to 4096 cores. Four 20 ϕ diodes per line are used for isolation, and two active current sources, at \$25 each, are employed for each 256-line matrix. The digit matrices, however, are more complex since their design must accommodate element sensing as well as selection.

In the contemplated scheme, lines are driven in parallel pairs to reduce "zero" noise and DC offset voltages which appear at sense-line terminals as a result of variations in core state, line resistance and diode drops (see Fig. 7). One drive transformer per pair of lines is included to allow matrixing of drive circuitry at one end of the line only. Lines are

shorted at the sink end but terminated at the source for the duration of the current pulse. Four diodes are provided at the sink end for each pair of lines to isolate unselected lines from the sensing circuitry and to provide a high-impedance path for rapid line and transformer recovery during turn-off of the transformer matrix. Two current-balancing transformers and one sense amplifier and filter may then service the entire digit matrix or a maximum of 256,000 cores. Sense-amplification and filtering functions were estimated at \$100 per matrix, and digit-selection components total to approximately \$300 for the largest matrix of 64 line pairs. Each of these lines threads 2048 cores, and maximum module size is then limited to $262,144 \times 32$. Cycle-time estimates of 3 μ sec are not unreasonable for this capacity and may be reduced by shortening the drive/sense wire to limit module size to a smaller number of addresses. Total memory-circuit cost, exclusive of logic, comes to 0.21 ϕ and 0.36 ϕ for $262,144 \times 32$ and $131,072 \times 32$ modules respectively, as shown in Fig. 8.

2D, Two-Wire Configuration

3D and $2\frac{1}{2}D$ organizations as applied to DRO systems require storage elements of sufficient quality to permit interrogation by the coincidence of two half-select currents. At present only ferrite cores are suitable, and those must be individually fabricated, tested, and wired into storage arrays. The property of the elements which permits coincident readout is simply the high ratio of outputs which occur for full- and half-select currents.

In a 2D, or linear-select, organization destructive readout occurs when current in a single wire switches all bits of the data word. Only one element per common sense-digit line experiences any excitation and, consequently, elements of much lower quality placed in arrays of two wires per element are suitable for comparable, and often superior, performance. In 2D organizations, however, element "squareness" is still a desirable property since a direct consequence is the percent of total element flux which may be modulated by a coincident write operation. In all cases, the digit field must be less than the element's coercive force so as not to disturb corresponding bits in other addresses. A square element where 90% of the flux may be modulated by a coincident write would give one/zero output ratios of the order of 10:1. On the other hand, a low-

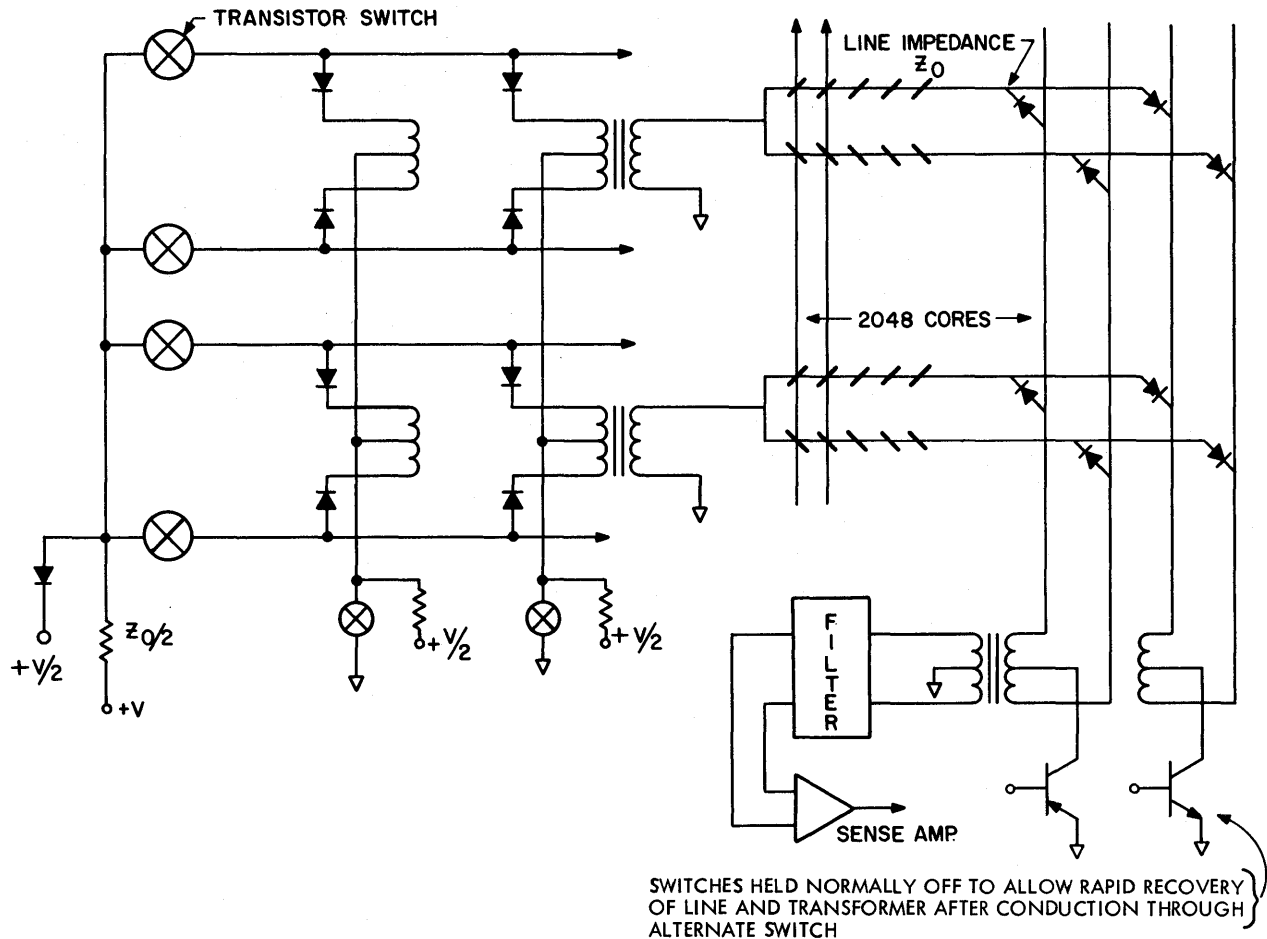


Figure 7. Digit matrix for $2\frac{1}{2}D$, two-wire scheme.

quality, low-cost element may be employed where only 10% of the total read flux is permanently affected by the digit field. In these cases, two intersections, or two elements per bit, are employed such that the bulk of the read flux cancels and the data-modulated portion adds to regain a distinguishable one/zero output-signal ratio.⁴

An advantage of linearly organized ferrite-core arrays is the accompanying ability to apply unlimited overdrive at read time and thereby increase switching speed over that available with coincident current selection.⁵ The write portion of the cycle, however, is necessarily a coincident operation, and one of the following three methods may be employed:

1. A full-select word current is overlapped by a positive or negative half-select digit current.
2. A full-select word current is overlapped by an opposing half-select digit current.

3. A half-select word current is applied in coincidence with a half-select digit current.

Method (1) yields a faster switching time with the added expense of full-select word drive circuitry and bipolar digit drivers. In addition, the bipolar digit requirement complicates the mixing circuitry which normally would combine digit and sense functions on a single wire. Relative economies of Methods (2) and (3) are obvious by consideration of the current amplitudes employed. An advantage of Method (2) however, is that elements are disturbed in the read direction only. "Zero" noise at readout is reduced from that which would occur with the opposite-polarity disturbs of Method (3). Use of elements with poor squareness ratios can make disturb polarity an important consideration. On the other hand, cycle time may be decreased with Method (3) since it is not necessary that digit currents overlap word currents. In any event, the prin-

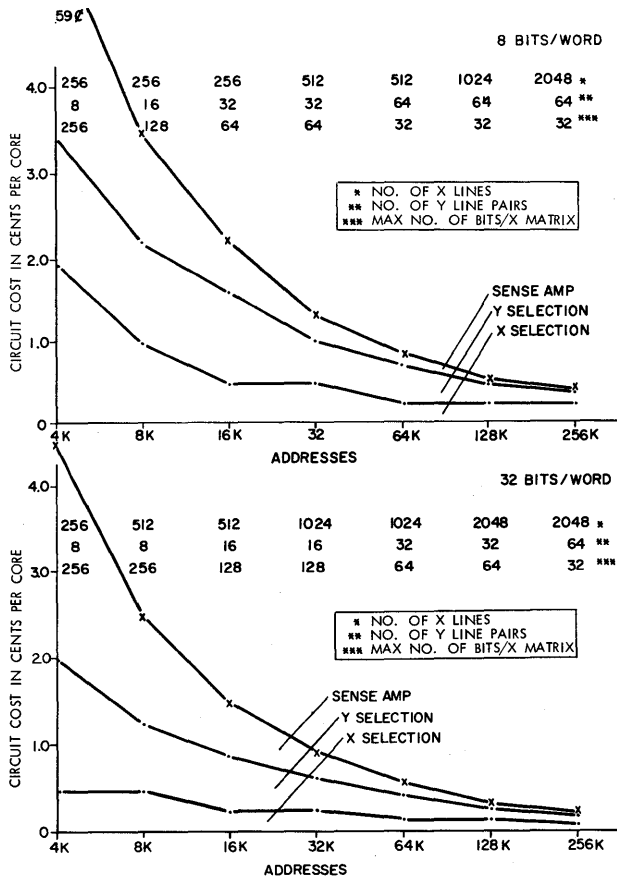


Figure 8. Circuit costs for optimum 2½D, two-wire configurations.

cial speed increase obtained by reorganizing a coincident array to operate in a linear select mode results from the decreased read switching time.

A more significant advantage is the available reduction of array complexity. Low-cost, two-wire arrays must be linearly organized to economically obtain speeds in excess of 1 μsec and, as memory speed requirements increase, necessitating the use of smaller toroids, third and fourth wire costs become increasingly significant in total array cost. In large 1-μsec systems for example, the cost of the fourth wire in 30-mil core arrays has already justified 2½D organization. Similarly the higher cost of the third wire in smaller 400- to 500-nsec core arrays may justify the next step in the array/circuit cost trade-off, i.e., a 2D linear selection scheme to either reduce a stringing operation to two wires per element or exploit the potential low-cost speed of various batch-fabrication techniques.

Increased circuit costs in a linear organization as opposed to coincident selection of similar storage

elements result from two factors. First, since word drive currents are destructive, digit circuitry cannot be matrixed to service a number of lines; second, the matrix word drive selection must handle full-select read currents. The inherent high circuit cost increases the effect of memory aspect ratio on total cost, and, lowest cost occurs in general, where word length is in excess of that normally required by a computer. In these cases, it is profitable to extend the memory data register to some multiple of the computer word length and provide address decoding to select the desired portion of the memory data word for read and write operations. Consider, for example, a system requirement of W words at B bits per word. This capacity may be realized by a memory with W_m addresses and B_m bits per word where

$$WB = W_m B_m$$

Total circuit and array cost per element is then

$$C = K_c + \frac{K_d}{W_m} + \frac{K_w}{B_m}$$

where K_c = cost per storage element,

K_w = cost of word drive circuits per word line,

K_d = cost of sense and digit circuits per regeneration loop,

and W_m exceeds the maximum selection-matrix size such that drive circuit cost per line is not a function of W_m .

The optimum aspect ratio may be determined by holding the product of W_m and B_m constant and setting the differential of total memory cost equal to zero. The lowest cost results when total word drive-circuit cost equals total digit- and sense-circuit cost. The ratio is then

$$\frac{W_m}{B_m} = \frac{K_d}{K_w}$$

Total cost for systems employing this optimum ratio then reduces to

$$C = K_c + 2 \sqrt{\frac{K_d K_w}{W \times B}}$$

where $W \times B$ is total memory capacity. The cost of added gating required when $B_m > B$ is considered negligible.

The relationship is, of course, invalid for systems where all word lines are selected with one drive matrix. For these systems, the cost of word drive

circuitry per line (K_w) is a function of W_m . In this case

$$K_w' = \frac{\text{total drive-circuit cost}}{W_m} = \frac{2K_s \sqrt{W_m} + D_1 W_m}{W_m}$$

where K_s = cost per switch or cost per switch pair for core systems requiring bipolar drive currents, and

D_1 = cost of the matrix component, i.e., diodes or transformers and diodes.

Substituting K_w' for K_w in the total cost expression gives

$$C = K_c + \frac{K_d}{W_m} + \frac{2K_s}{B_m W_m} + \frac{D_1}{B_m}$$

Total system cost is then optimized for the W_m , equaling some power of 2, which most nearly satisfies the following expression:

$$K_d = \frac{D_1 W_m^2}{W \times B} + \frac{K_s W_m^{3/2}}{W \times B}$$

To examine the cost characteristics of a linear organization, an example of a configuration is given which gives similar performance to previously discussed 2½D and 3D systems. Consider a low-drive, 30-mil core which would yield 1-μsec cycle times when linearly selected.

A 3/2 full-select read current of 750 ma and half-select 250-ma write current could be supplied by switch circuits similar to those used previously. In this configuration, circuit cost is increased by \$2.00 per switch pair over that used in other organizations to allow for high-current components. Three integrated diodes per line are provided to allow for the high read currents, and \$50 is included per matrix for two active current sources. Total cost of this configuration when selecting one of 256 lines amounts to \$606.

Only large systems are considered where word-selection circuitry is composed of numbers of these matrices. It follows that K_w is constant and equals $\$606/256 = \2.37 per line. Word lines are short, and back voltages would limit length to something less than 250 cores per line.

The digit drive, sense, and mixing circuitry must perform the same function as that described in the 3D, three-wire discussion. A similar configuration providing a full select current to be divided equally in two sense-line branches and balanced by a termi-

nation transformer could be provided at a cost of \$41 per bit and include a sense amplifier. These circuits may service a maximum of 4096 addresses, limiting module size to approximately 10^6 bits. Since $K_d = \$41.00$ for this configuration, the optimum ratio is

$$\frac{K_d}{K_w} = 17.3$$

Tabulating the total circuit costs for optimally organized systems of varying total capacity yields the figures shown in Table 2.

Table 2. Circuit Costs for Optimally Organized Systems of Varying Total Capacity

Words per Module (W)	Bits per Word (B)	Total Capacity in Bits ($W \times B$)	$\sqrt{\frac{2K_d K_w}{W \times B}}$
4096	236	0.965×10^6	\$0.02
2048	118	0.242×10^6	.04
1024	59	6.05×10^4	.08
512	30	1.53×10^4	.16
256	15	38.4×10^2	.32

This example is plotted in Fig. 9. The dotted line connects points where the best ratio of 17 is main-

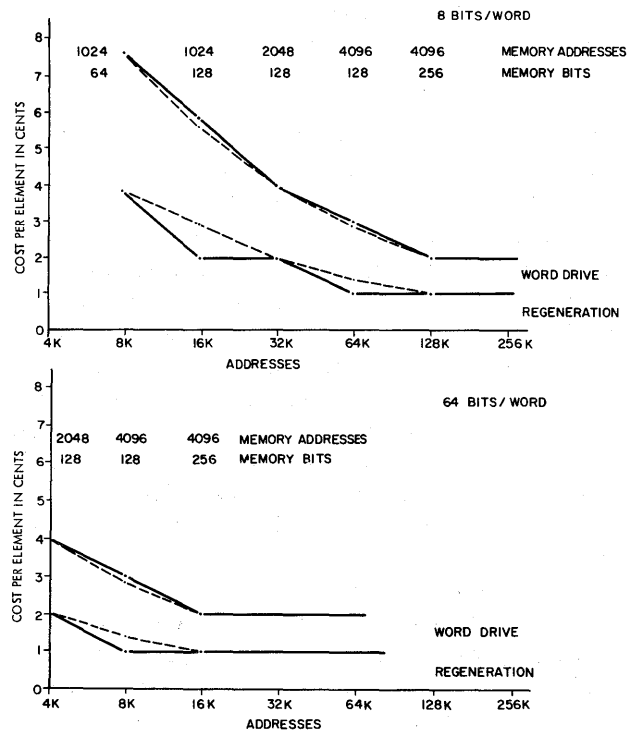


Figure 9. Circuit costs for optimum 2D, two-wire configurations.

tained. Practical solutions, however, are limited to those where the number of addresses equals some power of 2. These points are connected by solid lines and the resultant deviation is apparent.

COMPARISON OF RESULTS

In the preceding section, systems of varying organization were described briefly and cost characteristics were plotted individually to show the relative contributions of different circuit functions. These results are superimposed as shown in Figs. 10 and 11 to illustrate at what portion of the capacity spectrum each organization may be most effectively employed.

The 2½D, three-wire, and two 3D systems are the most similar of those discussed with regard to performance, circuits, core type, and applicable packaging techniques. The best range of capacity for these three systems is made apparent by the cost

crossovers shown. For systems requiring fewer than 2048 addresses, the four-wire approach shows slight improvement over the three-wire, 3D system, but vast gains may be achieved by selecting either 3D scheme over a 2½D design. The intersection of the 3D system at 2048 results from the increasing cost per element of the more complex sense-inhibit circuitry with decreasing capacity as opposed to the assumed constant cost of the fourth wire used in the alternate case. In some instances, however, the fourth wire cost would increase for smaller arrays. Although the cost of both systems is increased, the crossover would then move to lower capacities. As system size increases beyond 4096, however, sense-inhibit circuitry must be repeated for every 4096 addresses. The added per-element circuit cost of the three-wire scheme then becomes constant, and a cost differential of 0.3¢ per core results for all capacities in excess of 2048.

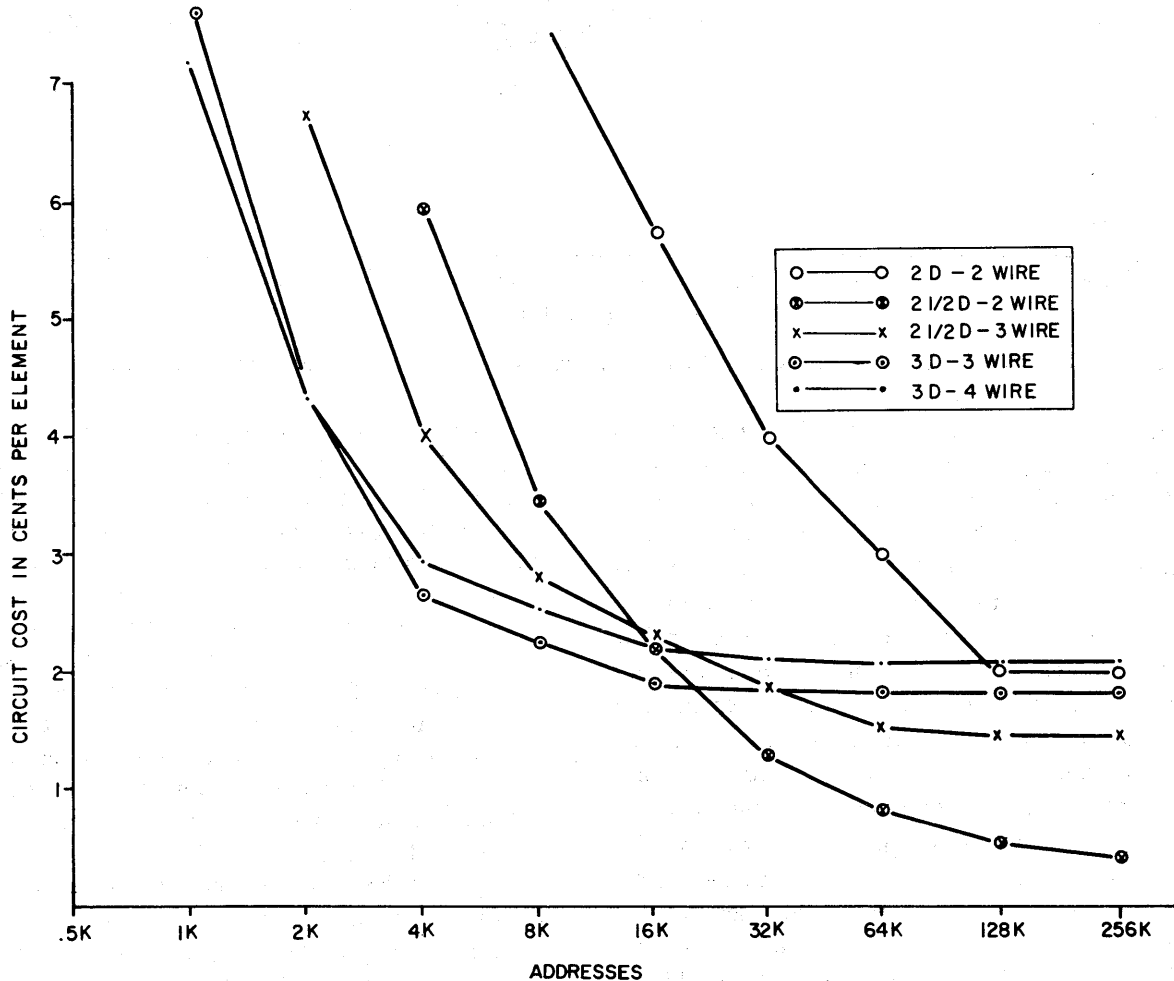


Figure 10. Drive and sensing circuit costs for various organizations as a function of memory capacity (at 8 bits/word).

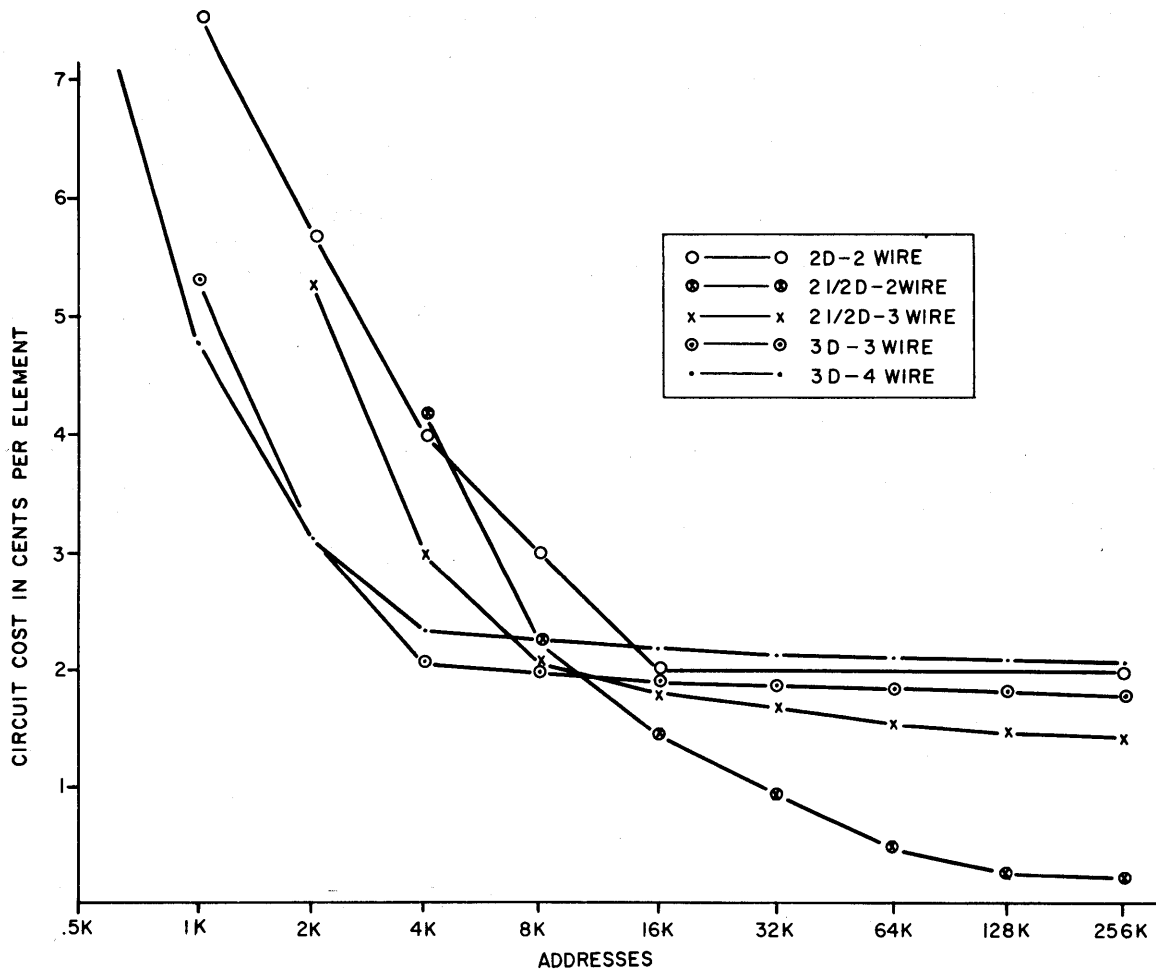


Figure 11. Drive and sensing circuit costs for various organizations as a function of memory capacity (at 64 bits/word).

The $2\frac{1}{2}D$, three-wire design becomes advantageous as address capacity increases to the range of 4096 to 16,384, depending on data word length. It, in fact, crosses the 3D, three-wire system since, in the $2\frac{1}{2}D$ scheme, selection circuits continue to increase with the square root of the number of addresses. In the 3D configuration, however, inhibit circuitry increases linearly with address capacity after 4096 and eventually involves a greater expense than the alternate three-wire arrangement. As capacity continues to increase, selection costs become approximately equivalent for all three approaches.

The $2\frac{1}{2}D$, two-wire design is plotted to show the savings which may be obtained in multi-megabit stores. Points at lower capacities represent unreasonable approaches since the cost is high and cycle times for any of these systems probably would exceed 2 μ sec. For addresses in excess of 32,768,

however, the approach may yield sufficient savings to justify a lower cycle time in certain applications. The cost difference here results from the vastly increased sensing efficiency. No sense line is required, and one detection circuit can service up to 262,144 cores.

The cost of the 2D organization is clearly high when compared to other medium-sized systems. Note, however, that the configuration competes for large systems, the difference being in the rate of cost increase as capacity decreases. A linear-select storage element might be obtained at slightly lower cost, and this would enhance the case for 2D at large capacities. For smaller sizes, the cost curve ascends quickly, and the array soon becomes an insignificant portion of the total. In general, the configuration is made popular only by the potential low cost and high speed of batch-fabricated arrays, nearly all of which must be interrogated linearly.

Effects of Increased Speed

With the exception of the mass memory, each design would provide cycle times of the order of 1 μ sec. Although actual costs were not plotted for higher speed systems, certain trends can be extrapolated from the data shown by understanding the nature of the circuit-array cost tradeoff. In the case of the three- and four-wire systems, each crossover is a result of selection circuit cost per element reducing as larger matrices become more efficient. This, compared with the constant cost of array complexity in the four-wire approach, results in the latter losing out at higher capacities. As speed requirements increase, however, compatible storage arrays will become more costly and probably at a more rapid rate than corresponding circuit increases. In this event, third and fourth wire costs will exceed the half-cent figure used in these plots for 30-mil cores, and the crossovers will move toward lower capacities as core diameter is necessarily reduced. The 3D, three-wire approach, for example, would be superior to a four-wire configuration for all capacities if core size were reduced to 20 mils or less. Similarly, linear-select systems will become practical for smaller capacities as core diameter is reduced and will eventually win out over all alternatives as cycle times fall below 500 nsec. The crossover of the three-wire designs would be less affected since similar arrays are used. Discontinuities in cost/speed curves here, however, would show the 2½D design to be superior since better cycle times (15%) may be achieved for a given core size with this approach.

Selection vs Sensing

The curves in the preceding section show the relative costs of various circuit functions for systems of each organization. It is interesting to note that, in all designs other than the mass store, cost reduction is limited by sensing and inhibit circuitry which repeats for every 4096 cores. In this range, cost of selection circuitry is almost insignificant, and changes in sense amplifier or array prices can reflect as large percent changes in the total. Consequently, improved sensing efficiency is the most fruitful area of study for cost reduction in 2½D systems of 8192 and larger. Reducing selection circuitry cost is effective in 2½D, three-wire machines containing less than 32,768 addresses. Larger 2½D and 3D systems would benefit to a lesser degree.

CONCLUSIONS

In summary, a number of remarks may be made:

1. The 3D approach at present is the most economical design for systems of 8192 addresses or less when required cycle rates do not exceed 1 MHz. At some added cost, however, better cycle times may be achieved in this capacity range by selecting a 2½D, three-wire design using the same 30-mil core. This approach would often be cheaper than maintaining the 3D configuration with a smaller element.
2. Of the two 3D designs, the three-wire approach is favored for large systems, although the crossover is extremely dependent upon the fourth-wire cost in the storage array.
3. The 2½D, three-wire design is best for systems larger than 8192 when speeds higher than 3- μ sec rates are needed. For lower speeds, a 2½D, two-wire scheme, may be best employed.
4. As speed requirements increase, necessitating the use of smaller elements, crossovers will occur at lower capacities. Consequently, 2D and 2½D designs using simpler arrays will become practical for smaller systems as well as large.

ACKNOWLEDGMENTS

The author would like to acknowledge the assistance of G. W. Booth, R. W. Reichard, W. F. Jordan and R. W. Fletcher of Honeywell Computer Control Division in preparing this material.

REFERENCES

1. J. W. Forester, "Digital Information in Three Dimensions Using Ferrite Cores," *J. Appl. Phys.*, vol. 22, p. 44 (1951).
2. M. A. Alexander, M. Rosenberg, and R. Stuart-Williams, "Ferrite Core Memory is Fast and Reliable," *Electronics* vol. 29, no. 2 (Feb. 1956).
3. R. J. Petschauer, G. A. Andersen, and W. J. Neumann, "A Large-Capacity, Low-Cost Core Memory," presented at IFIP Congress, New York, May 24-29, 1965.

4. R. A. Shahbender, et al, "Laminated Ferrite-Memory," *Proc. Fall Joint Computer Conf.*, 1963.
5. C. S. Holzinger, "Techniques for Determining the Speed Capabilities of 2D Ferrite Core Memories," *IEEE Trans. on Magnetics*, vol. 1, no. 4 (Dec. 1965).

A 200-NANOSECOND THIN FILM MAIN MEMORY SYSTEM

S. A. Meddaugh and K. L. Pearson

*UNIVAC Division of Sperry Rand Corporation
St. Paul, Minnesota*

INTRODUCTION

Several papers have appeared in the last few years which propose a design for large, high-speed memories using planar thin films. Included in this category are memories with greater than 250,000 bits and cycle times of less than 250 nanoseconds. Some authors^{1,2} have set rather high goals of 10^6 bits and 100 nanoseconds cycle time, and, after performing a number of calculations, have concluded that it is indeed possible for such a memory to operate, provided the problems of building it can be solved. Others³ have presented the results of early, partially implemented models with less ambitious goals, and of course have concluded that a full-sized memory is indeed feasible. These are necessary steps preceding the building of a fully populated, reliable, manufacturable memory. This paper describes the design of such a memory.

Capacity of this memory is 4096 68-bit words (278,528 bits, to be exact) and it operates with a cycle time of 200 nanoseconds and an access time of 160 nanoseconds. It is a word-organized, random-access memory. The memory element is composed of a pair of planar thin films coupled together and read out destructively.

As the memory is intended for a military application, Mil-Specs were observed and worst-case design was used.

SYSTEM ORGANIZATION

A large, fast memory requires parallel organization primarily to avoid the accumulation of excessive digit line delay. Thus, this memory has been organized into four 1024-word stacks, each with its own digit drivers and sense preamplifiers. The remainder of the digit circuitry is shared between stacks and is located in the two circuit chassis. Refer to Fig. 1 for the block diagram of this configuration and to Figs. 2 and 3 for the physical arrangement.

The word access system must be divided into small

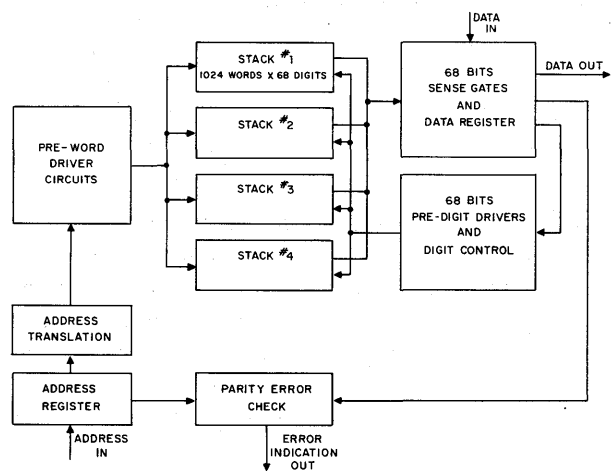


Figure 1. Basic memory system organization.

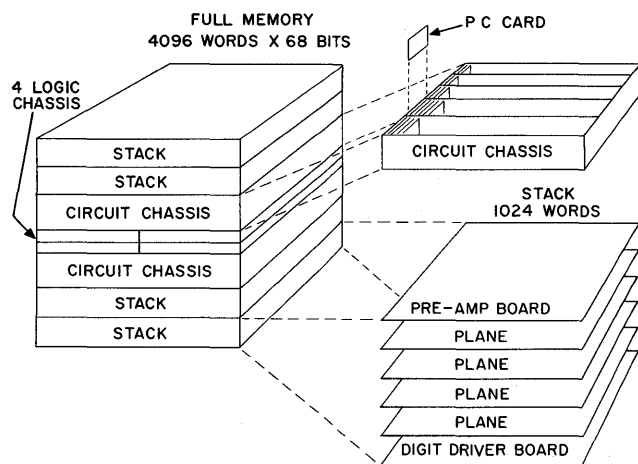


Figure 2. Physical arrangement of memory.

matrices to minimize selection delays. Here the matrix size is 16×16 , with each matrix, along with the output stage of the associated word drivers, contained on the front half of the memory plane. Pre-word driver circuits are located on the circuit chassis.

The address register, address translation, parity error checking circuits for both address and data, and timing and control circuits are all located in the four logic chassis.

The four stacks and two circuit chassis are identical and interchangeable among themselves.

STORAGE ELEMENT

The storage element utilized in this memory consists of two planar magnetic thin films operating in a coupled mode. The films are made of nickel-iron, vapor-deposited through masks on 3-mil glass substrates to form 30-mil square bits. Typical characteristics⁴ of the uncoupled films are:

Thickness	1100 angstroms
Anisotropy field (H_k)	4 oersteds
Coercivity (H_c)	2 oersteds
Skew (β)	2 degrees
dispersion ($\propto \omega_0$)	4 degrees

Two films are coupled together around drive and sense lines to form a single element, as shown in Fig. 4. Thus, the field applied to the top and bottom films, by current flowing in both word and digit lines, is in opposite directions, so that the magnetization vector in the two films is always antiparallel. Therefore, the two films are always coupled, regardless of applied field.

The benefits of coupling films in this manner are modified by the distance the films are separated by

the conductor array. The array is constructed of $\frac{1}{2}$ -mil copper bonded to both sides of $\frac{1}{2}$ -mil polyester film and etched to form 20-mil-wide word and digit lines. Film core arrays are then bonded to both sides of the array. The resultant spacing between films is about 2 mils, which is sufficiently close to improve film characteristics significantly over uncoupled films.

The resultant coupled-film characteristics are shown in Fig. 5. The digit curve, Fig. 5a, is a plot of output available for a given digit current, for a single write, after an adverse history, followed by many disturbs.

Output is essentially constant from 100 ma to 300 ma for digit-disturb only; when a 30-ma word-disturb pulse is added, the upper limit is reduced to 200 ma. The maximum word-disturb current the films will have to withstand is less than 30 ma.

The word plot, Fig. 5b, shows undisturbed film output as a function of word current. While the flux output levels off at about 600 ma, peak output amplitude increases beyond this point, since the films switch faster for large currents as the effective rise time to 600 ma is less.

From these plots and other considerations, minimum word current was chosen to be 700 ma, and digit current 140 ± 20 ma.

WORD AND DIGIT LINES

The geometry of the word and digit lines in the neighborhood of the films is shown in Fig. 4. Not shown are the top and bottom ground planes which are spaced about 3 mils away from the films by the glass substrates.

Both word and digit line are slotted. This was found to be necessary in order to allow the films to switch completely from, for instance, a one to a zero during a writing operation with a word current pulse width of 40 nanoseconds. Unslotted lines cause flux to be trapped, particularly in the word line, causing a restoring torque to be applied to the films. One slot in the digit line and two in the word line were found to be sufficient.

The word line is 68 digits long, has a Z_0 of 16 ohms and a delay of 2 nanoseconds and is terminated as shown in Fig. 6a. A far-end short-circuit termination was chosen to minimize the word line voltage swing to prevent excessive word noise caused by unbalanced capacitive coupling to the two legs of the digit line. A shunt-resistive termination at the driving end is necessary to minimize reflections.

The digit line is used both for driving digit current

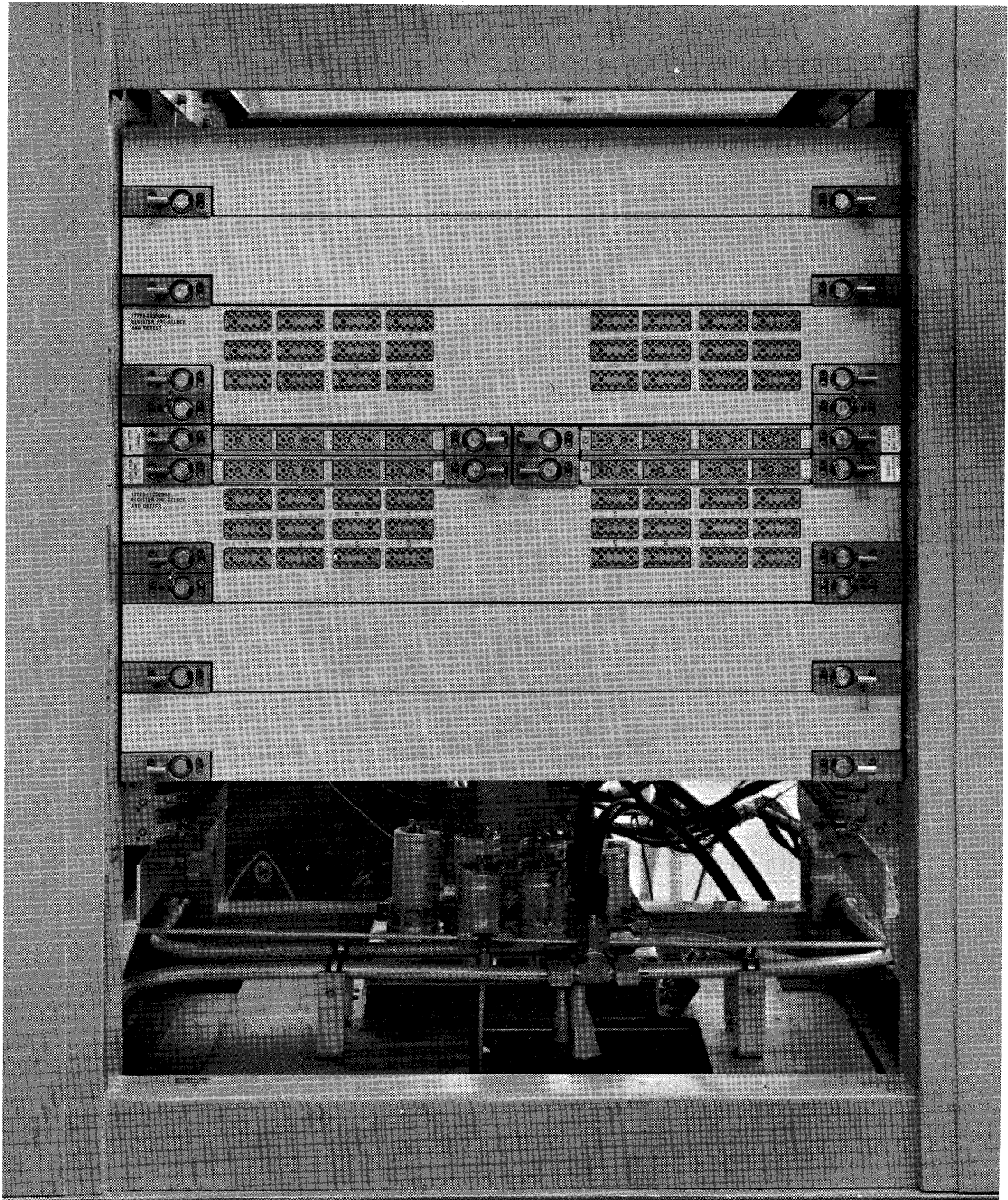


Figure 3. 200-nanosecond memory—4096 words, 68 bits.

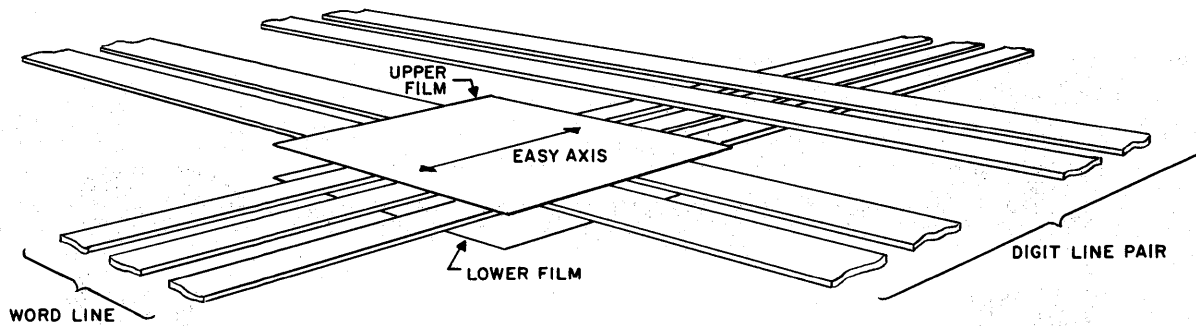


Figure 4. Memory element geometry. Not shown are the surrounding ground planes.

and sensing film output. Experimentally, this was found to yield much lower digit noise than the alternative separate sense and digit lines. Capacitive and inductive coupling between the separate lines is very strong in this system, and thus, the sense and digit lines must not only be well balanced to ground but to each other—a more difficult task than balancing a single digit line to ground. In addition, the more complicated connection scheme for the separate sense and drive lines is a source of digit noise. Good resistive balance of the common sense-digit line is not difficult to achieve since all connections are soldered. Furthermore, the elimination of one layer of etched wiring allows the films to be more tightly coupled and simplifies construction problems.

The length of the digit line is limited to 1024 words, to keep the delay within reasonable limits. Z_0 is 12 ohms each leg to ground and the delay for 1024 words is 16 nanoseconds. The termination networks, shown in Fig. 6b, are designed to allow difference and common-mode signals to be terminated separately.

Since the down-and-back delay of the digit line of 32 nanoseconds is much longer than the 10 nanosecond digit current risetime, it is necessary to exactly terminate digit current (a common-mode signal) at the right end of the line. Film signal is 10 nanoseconds wide—again much shorter than the down-and-back delay—making it desirable to terminate the backward-flowing portion of film signal at the left end. This prevents it from appearing at the sense pre-amplifier at varying times with respect to the forward-flowing film signal, depending on the film's position along the digit line. Line attenuation is 2 db.

The exact difference termination on the left end helps reduce noise coupled from adjacent digit lines by terminating backward-coupled noise instead of allowing it to propagate back to the sense amplifier, as in the shorted case. Forward-coupled digit noise is

reduced by the crossovers; however, the crossovers are not close enough together to have much effect on backward-coupled noise. Only the two adjacent digit lines couple any noticeable noise, with the worst-case total being about three times the signal amplitude. Self-digit noise is typically equal to the minimum signal amplitude.

All connections in the digit line are made with micro-strip line soldered in place to maintain uniform line characteristics and balance through the connection area.

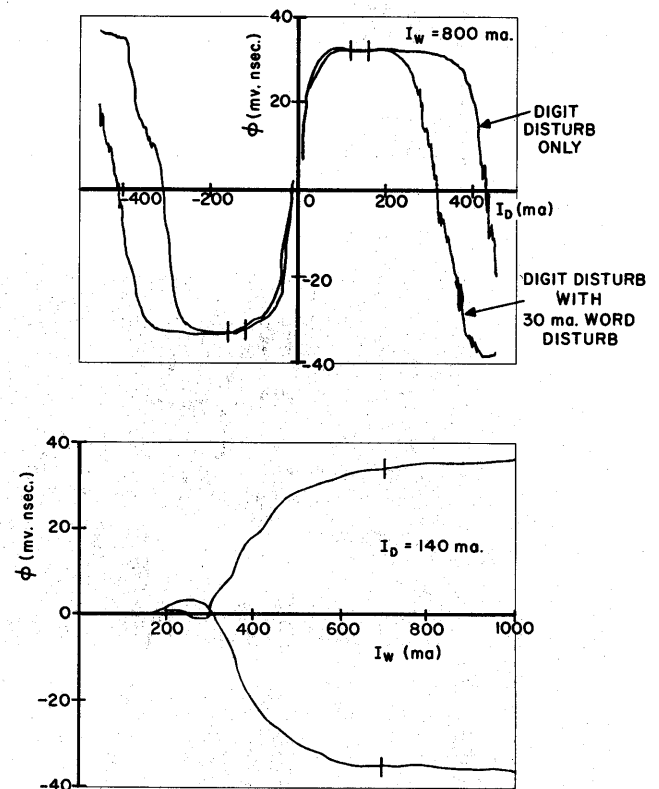


Figure 5. Typical film output vs digit current (a), vs word current (b).

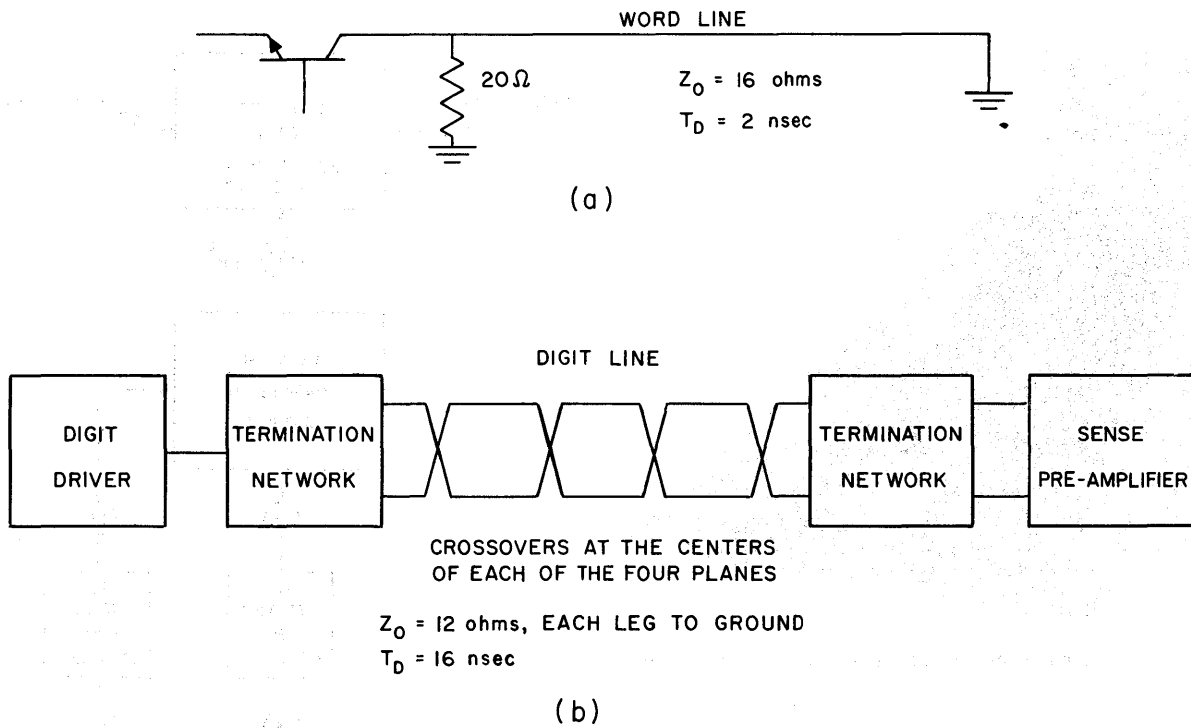


Figure 6. Word and digit line characteristics and terminations.

STACK ORGANIZATION

A 1024-word stack contains four identical memory planes consisting of two sealed memory element packets each containing 128 68-digit words, and a word access board containing the final stages of the word access circuits. Also included in the stack are sense preamplifier and digit driver boards, each con-

taining 68 circuits and digit line terminations. The physical arrangement of these planes into a stack is shown in Fig. 2.

The flexible strip line connections between planes allow the planes to be hinged together to permit unfolding the stack to make repairs and to facilitate testing. The stack appears in its folded and unfolded states in Figs. 7 and 8.

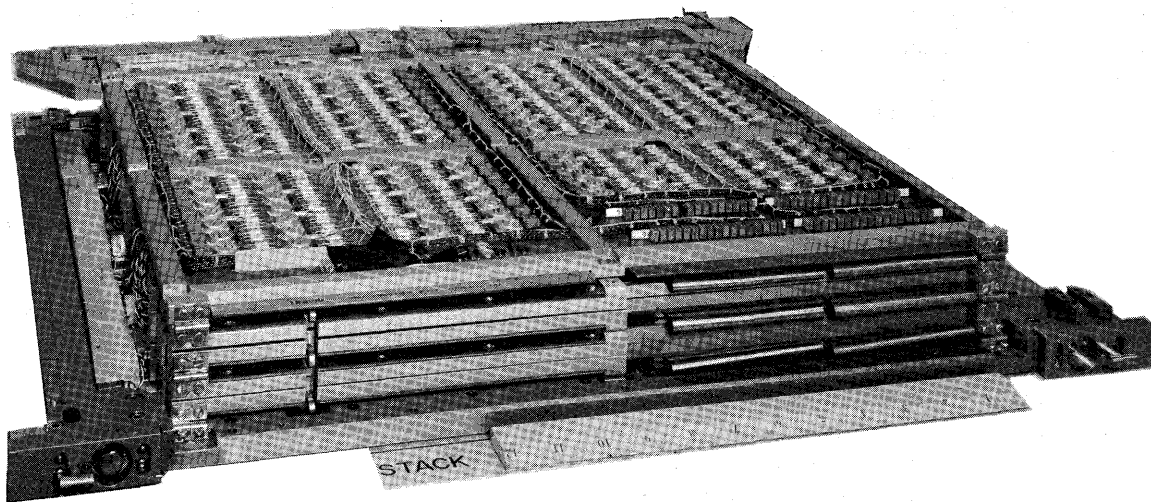


Figure 7. Folded stack. Sense preamplifier board is on top.

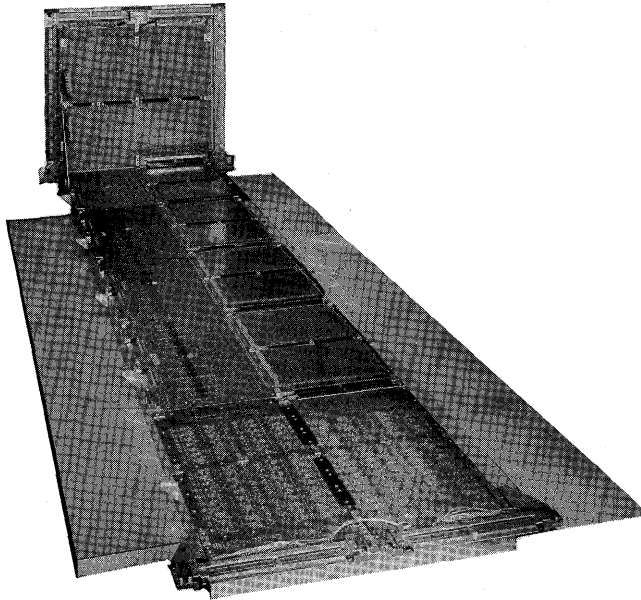


Figure 8. Unfolded stack. Sense preamplifier board is at near end.

WORD ACCESS CIRCUITS

The word-access matrix in this memory employs transistors as the selection element. Selection is performed on the base-emitter junction of the transistor in a manner similar to that used in a diode-access matrix. Unlike conventional diode-matrices, the transistors allow the half-selected word lines to be isolated from the access-matrix resulting in lower word noise and lower word line sneak currents.

A block diagram of the portion of the word selection system contained on a 256-word plane is shown in Fig. 9. The word-access matrix is composed of a transistor-per-word organized in a 16×16 array located immediately adjacent to the memory element packet. The bases of the word driver transistors are connected together in rows and the emitters are connected together in columns. Selection of one base line, followed by the selection of one emitter line, causes the word driver transistor at the intersection of these lines to turn on, thus allowing word current to flow down the associated word line.

The schematic of the word selection array is shown in Fig. 10. The word driver transistors are packaged four-per-integrated-package (a 0.6 inch diameter, 11 pin header) with the emitters common. The word line termination resistor is integrated in the same package. The word driver transistors are operated in a linear common-base mode driving a word line with a delay of two nanoseconds and a characteristic im-

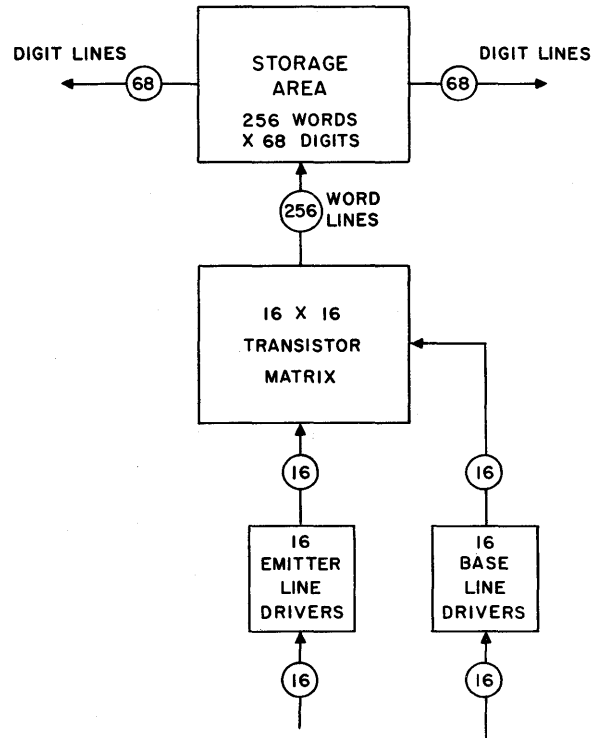


Figure 9. Memory plane organization.

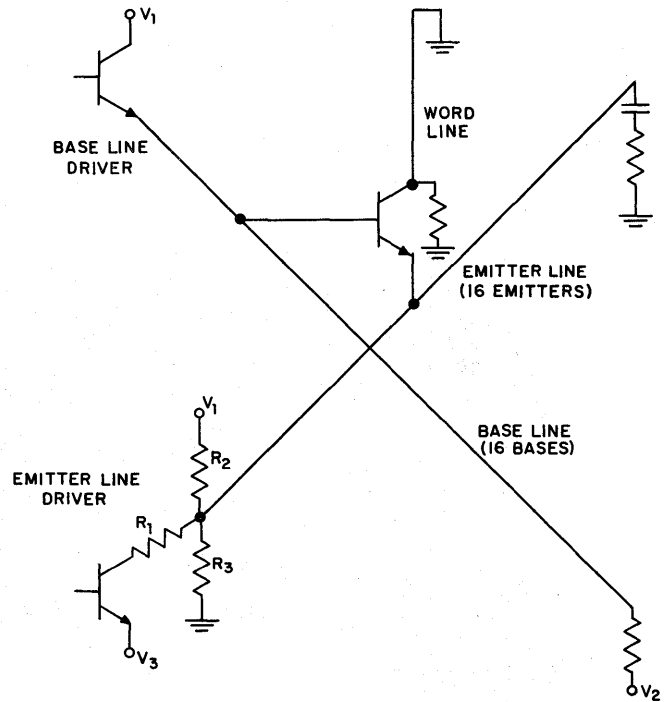


Figure 10. Schematic of 16×16 word selection array.

pedance of 16 ohms, shorted at the far end and shunt-terminated in a resistance slightly larger than Z_0 at the near end. Rise and fall times of the word current are 8 nanoseconds and the driving voltage is about 4 volts as shown in Fig. 11 for a nominal word current of 800 ma. The word drivers are turned on for read, turned off, and turned on again for write to limit power dissipation in the system components.

The base lines have a characteristic impedance of 10 ohms and are terminated at one end in Z_0 and at the other end with a saturated transistor. Thus, the impedance seen by the base of the word driver transistor is less than 5 ohms. The base line delay through the array is 7 nanoseconds. Base line return voltage, V_2 , is regulated and referenced to track V_1 to prevent excessive base-emitter reverse bias on the word driver transistors.

The emitter lines have a characteristic impedance of 5 ohms and are terminated, during the rise time, with a series RC network. Line delay across the array is 1.5 nanoseconds. The emitter line drivers are turned on after the base line is selected and stable. The negative excursion of the emitter line driver collector voltage forward biases the selected word-driver transistor permitting word current to flow. The amplitude of the word current is fixed by V_1 , V_3 , and R_1 . V_3 is regulated and referenced to track V_1 , to prevent voltage tolerance buildup from affecting current amplitude. A program-controlled adjustment is included in this regulator to permit variation of word current $\pm 10\%$ of nominal for on-line margin checking. Timing inputs to the emitter line drivers are incremented in four steps of 4 nanoseconds, each,

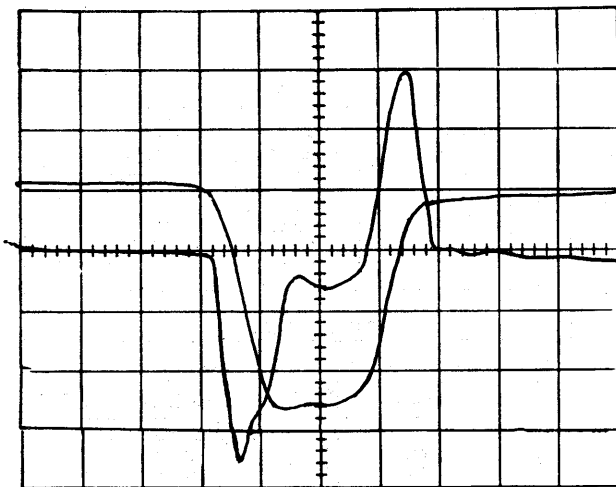


Figure 11. Word line waveforms, 10 nsec/div. Top: Word line current, 200 ma/div. Bottom: Word line voltage, 1 volt/div.

to compensate for the digit line delay. The maximum time from the initiate to establish word current is 75 nanoseconds.

All transistors in the selection system are normally off, resulting in low idle power.

The emitter line drivers are mounted directly behind the transistor array to permit short emitter selection lines. The less critical base line drivers are at the rear of the plane with the line termination resistors located at the sides of the array. Alternate base lines are driven from opposite sides of the array to facilitate packaging. Base line and emitter line pre-drivers to provide amplification from logic level to access plane current requirements are mounted on printed circuit cards located in the circuit chassis, a photograph of which is shown in Fig. 12.

DIGIT SYSTEM

A block diagram of the digit system is shown in Fig. 13. The line terminations, sense preamplifiers, strobe preamplifiers, digit drivers, and digit timing circuits are mounted in the stack on the upper and lower planes. All other circuits are located in the circuit chassis.

The integrated sense preamplifier is a two-stage, differential, emitter compensated amplifier with the stages capacitor-coupled. The two stages are identical and packaged, one stage per integrated package. The emitter resistors are discrete components and the resistor in the second stage is selected in five steps to adjust midband gain. For each value of this emitter resistor, a corresponding value of emitter-compensation-capacitor is used in each stage to adjust the frequency response. The nominal midband voltage gain is 70 into a 140-ohm load with a rise time of 7 nanoseconds. A typical sense preamplifier output is shown in Fig. 14.

The sense preamplifier outputs of the four stacks are OR'ed together at the sense gate in the circuit chassis. The interconnecting wire is 140-ohm twisted-pair terminated in Z_0 with a series resistor in each side and a low-input, impedance-differential, common-base stage. The resultant OR efficiency is about 90% at the cost of eight interface pins per bit.

The sense gate is composed of a common-base, differential input stage capacitor coupled to a common-emitter differential stage with emitter feedback. DC restoration is provided by a centertapped 15-nanosecond delay line connected between the bases of the common-emitter stage with the centertap re-

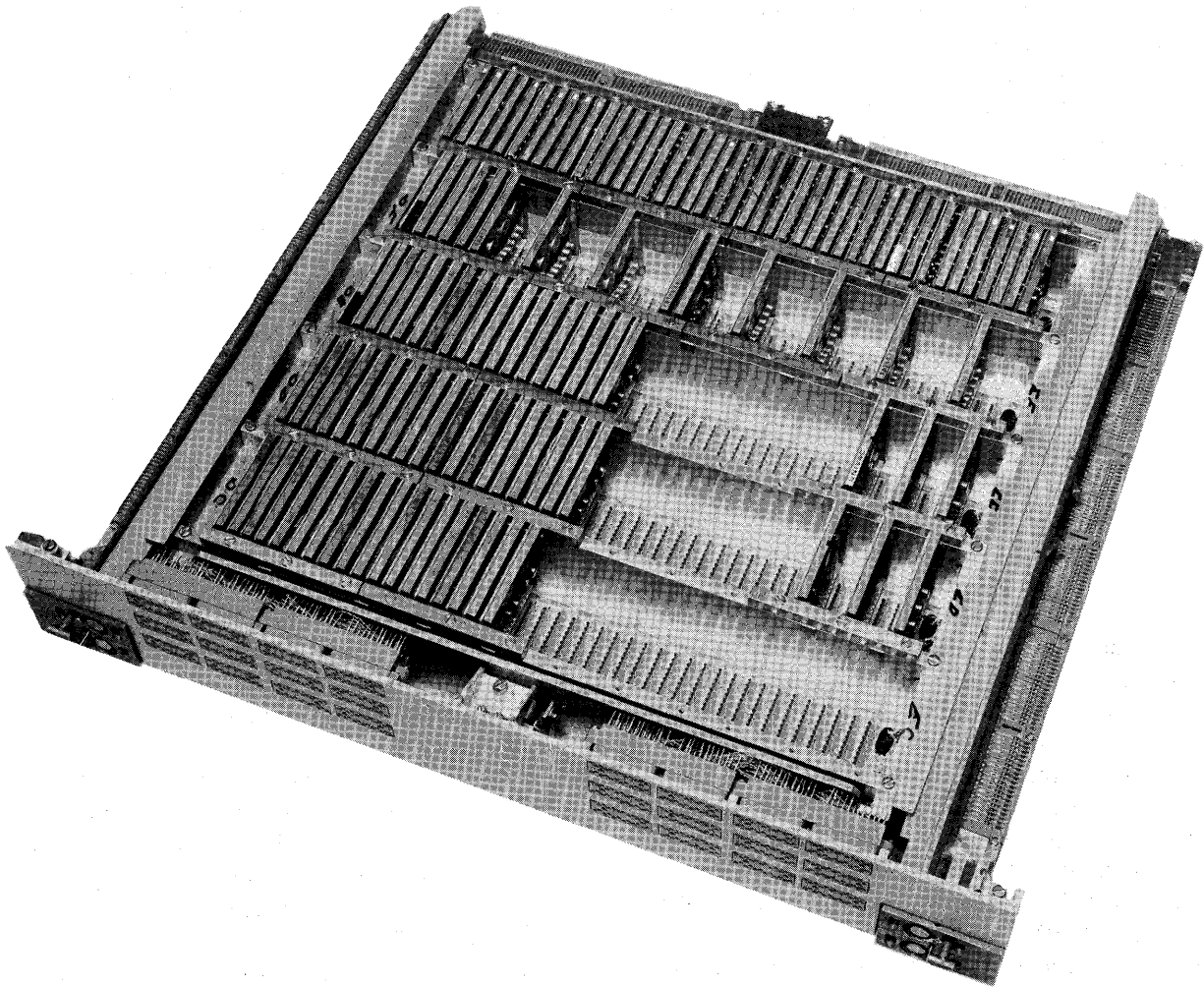


Figure 12. Circuit chassis with printed circuit cards.

turned to a threshold voltage. A block diagram is shown in Fig. 15.

Direct-coupled to the collectors of the second stage is a modified-current-mode flip-flop which serves as the sense gate latch and data register.

This circuit has three stable states and, when cleared, requires a positive signal to set to the one state and a negative signal to set to the zero state. The output of this circuit is shown in Fig. 16. The latch is forced clear before signal time with a current pulse. The removal of this current pulse serves as the strobe which allows the latch to be steered with signal. To ensure that noise does not set the latch, the strobe is enabled during the usable signal duration of about 10 nanoseconds.

The critical timing of the strobe pulse is achieved

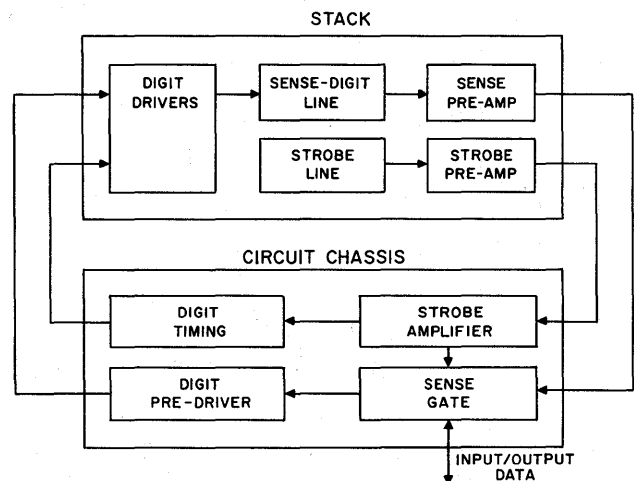


Figure 13. Block diagram of sense-digit system.

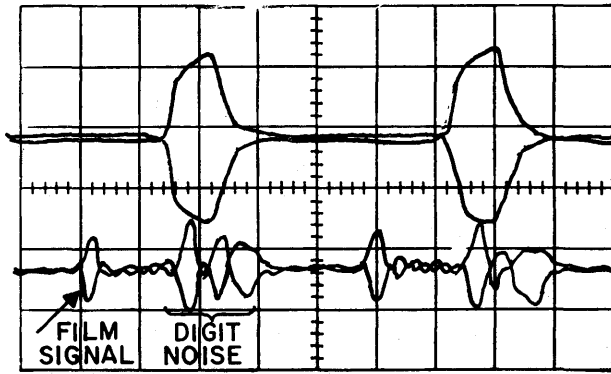


Figure 14. Digit line waveforms, 40 nsec/div. Top: Digit current, 100 ma/div. Bottom: Sense preamplifier output, 200 mv/div.

by means of an access-derived enable. A strobe line with a delay identical to the digit line is located orthogonal to the emitter lines and parallel to the

determined by the emitter current of the latch, which is dependent on three system voltages. To make this current constant, the threshold-voltage reference at the delay line is derived with a temperature-compensated regulator that tracks the three critical voltages simultaneously.

Writing of external data into the memory is accomplished by forcing the latch to the one or zero state prior to the sense signal being received from the read portion of the cycle.

The output of the data register, shown in Fig. 17, is sent to the external system through a line driver, to logic for parity checking, and to the digit predrivers. The digit predriver, together with an even-odd address decoder, determines which polarity digit driver will be enabled. This enable is fanned-out to a digit driver in each of the four stacks but only one

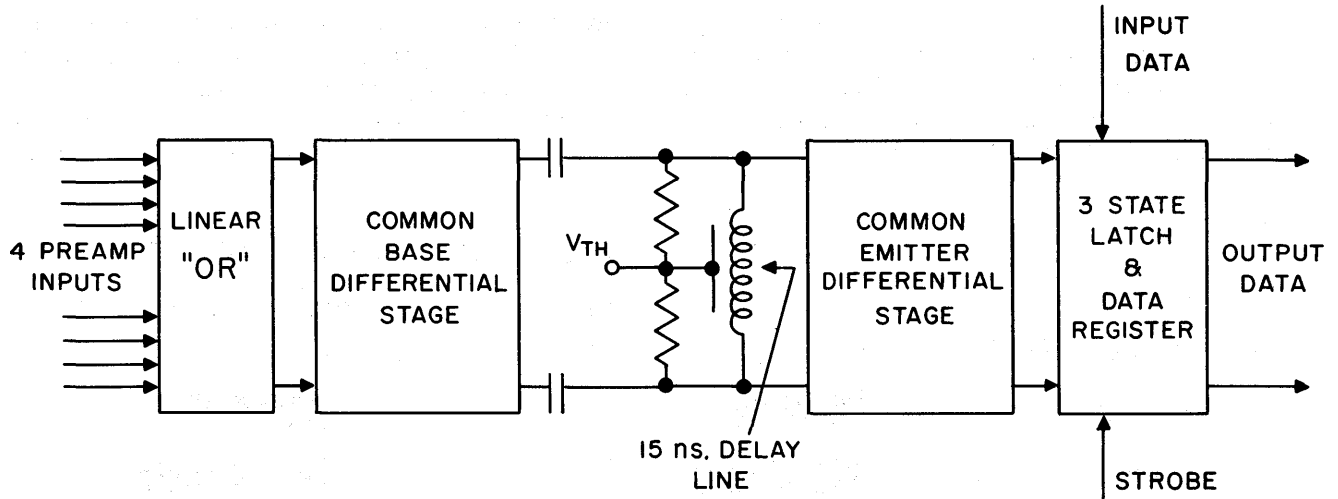


Figure 15. Block diagram of sense gate.

digit lines at the collectors of the emitter line drivers. The capacitance at each junction is approximately 2 pf, and a nominal signal of 60 mv is induced in the line when an emitter line driver is turned on. The strobe line location at the emitter line driver output, rather than using a dummy sense line, was chosen because an order of magnitude greater signal is obtained 5 nanoseconds ahead of the sense signal at the cost of a slight additional timing jitter of the emitter-line delay and word-driver delay variations. The strobe signal is amplified on the preamplifier plane and OR'ed at the circuit chassis in a manner similar to that of the sense signal.

The amplitude threshold of the three-state latch is

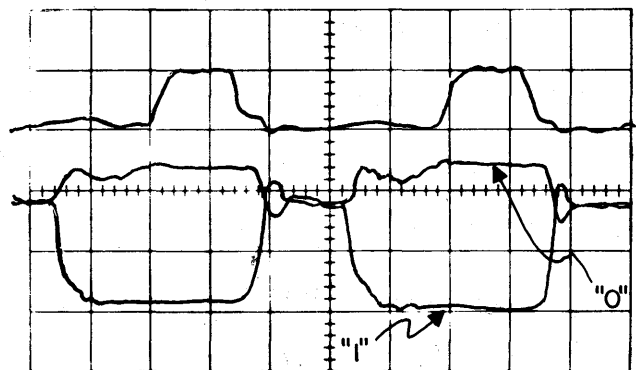


Figure 16. Three-state latch output referenced to initiate, 40 nsec/div. Top: Initiate pulse, 2 volts/div. Bottom: Three-state latch output, 2 volts/div.

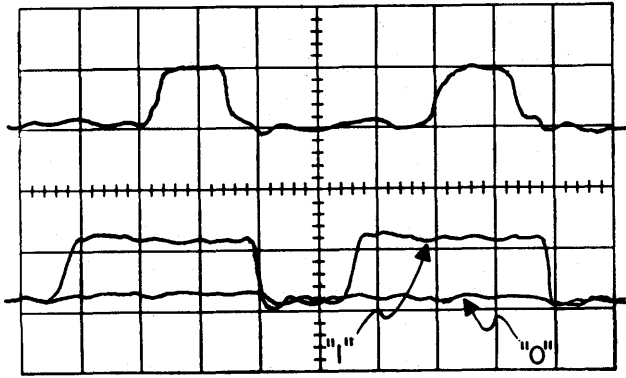


Figure 17. Output data referenced to initiate, 40 nsec/div. Top: Initiate pulse, 2 volts/div. Bottom: Output data, 2 volts/ div.

stack receives timing. Digit timing is enabled by the access-derived strobe pulse so the word-current digit-current relationship is constant.

The digit driver is a bipolar, transformer-coupled, two-stage switch direct-coupled to the digit line. The nominal digit current is 140 ma (70 ma per line) with a 10-nanosecond rise time and a 15-nanosecond fall time as shown in Fig. 14. Total sense-digit loop-delay, including the digit line, is 60 nanoseconds.

LOGIC

The standard logic module is an integrated package with resistor-transistor logic. Typical switching speed is 4 nanoseconds and average propagation delay is approximately 5 nanoseconds. The logic modules are mounted on a logic chassis with space for 297 modules. Intrachassis wiring utilizes wire-wrap connections.

Main timing is accomplished with delay lines located on the logic chassis with 5-nanosecond taps for rough timing and 2-nanosecond taps for fine timing. A diagram of significant memory timing, indicating worst-case anticipated logic and circuit timing jitter, is shown in Fig. 18.

SYSTEM FEATURES

The use of four identical substack assemblies and two identical circuit chassis assemblies in the makeup of the memory permits complete interchangeability of these units and greatly reduces spare requirements. In addition, this building block organization permits various other system arrangements to be imple-

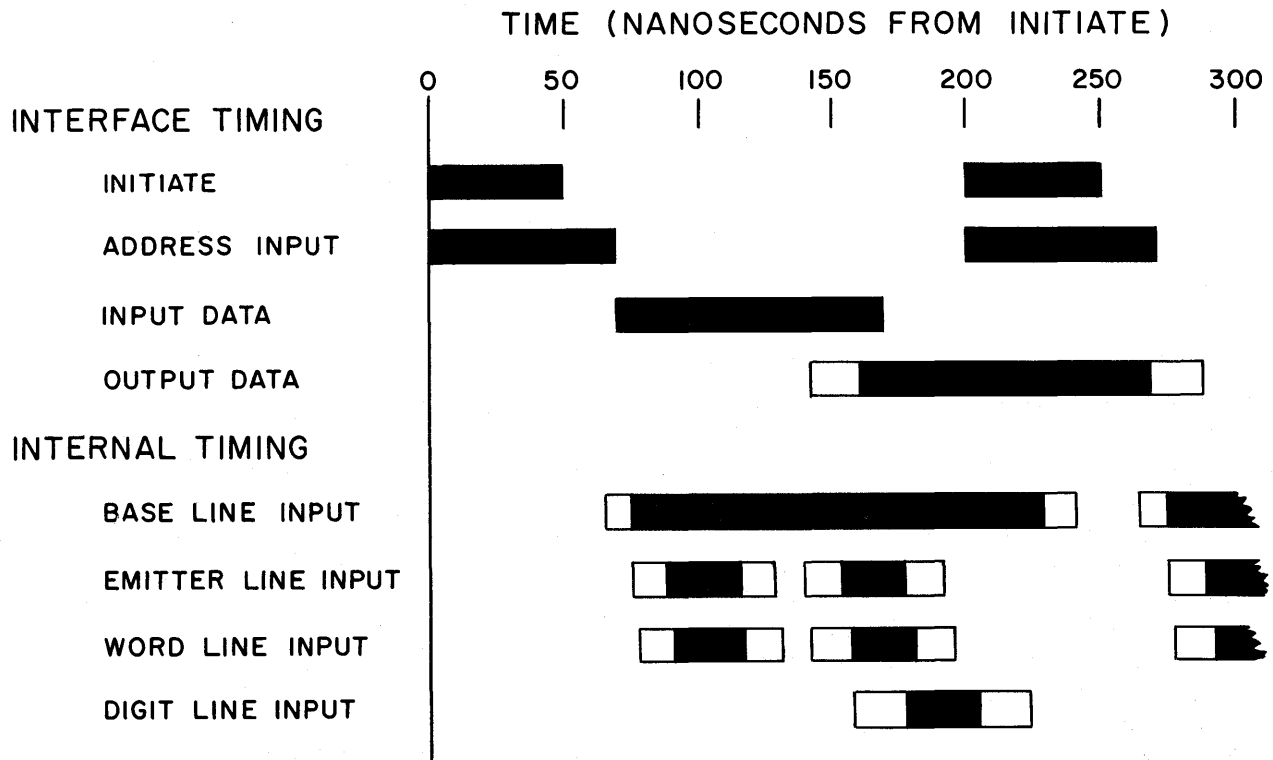


Figure 18. Memory timing diagram.

mented. Memories of 1024 words, with either 34 or 68 bits, require no changes in the memory hardware.

The 68-bit data word may be used by the external system as two 34-bit words. A byte-select bit in the address register permits writing new data in one 34-bit byte while restoring the other 34 bits. Parity is checked independently in the two bytes. Thus, the memory can be used as an 8192 word, 34-bit memory.

Indicator drivers are provided on all registers and control flip-flops, and may be used by the external system to indicate memory contents. The registers are held "set" at the end of the cycle to facilitate indication. Error detection is accomplished by means of parity checking.

PERFORMANCE

Performance data presented in this section was obtained from a developmental model, which is essentially the same as the production models, except that two stacks are omitted for reasons of economy. The

remaining two stacks can be placed in any of the four stack positions without special tuning and performance will not be affected.

Data on margins is obtained by running the memory with an exerciser which is capable of generating a number of different patterns to test for such things as base line shift in the sense amplifier, element disturb, and others. Using the full repertoire of tests, voltage margins are greater than $\pm 5\%$ with the memory operating at 200 nanoseconds. Cycle time, likewise, can be decreased more than 5% from nominal. Figure 19 shows a word and digit current schmoo plot for a 1024-word stack operating at 200 nanoseconds.

As mentioned, the exerciser contains a very comprehensive disturb testing mode. No disturb problems have been encountered in the stacks used in this model.

SUMMARY

The memory design presented in this paper has been shown to be a good workable design which is

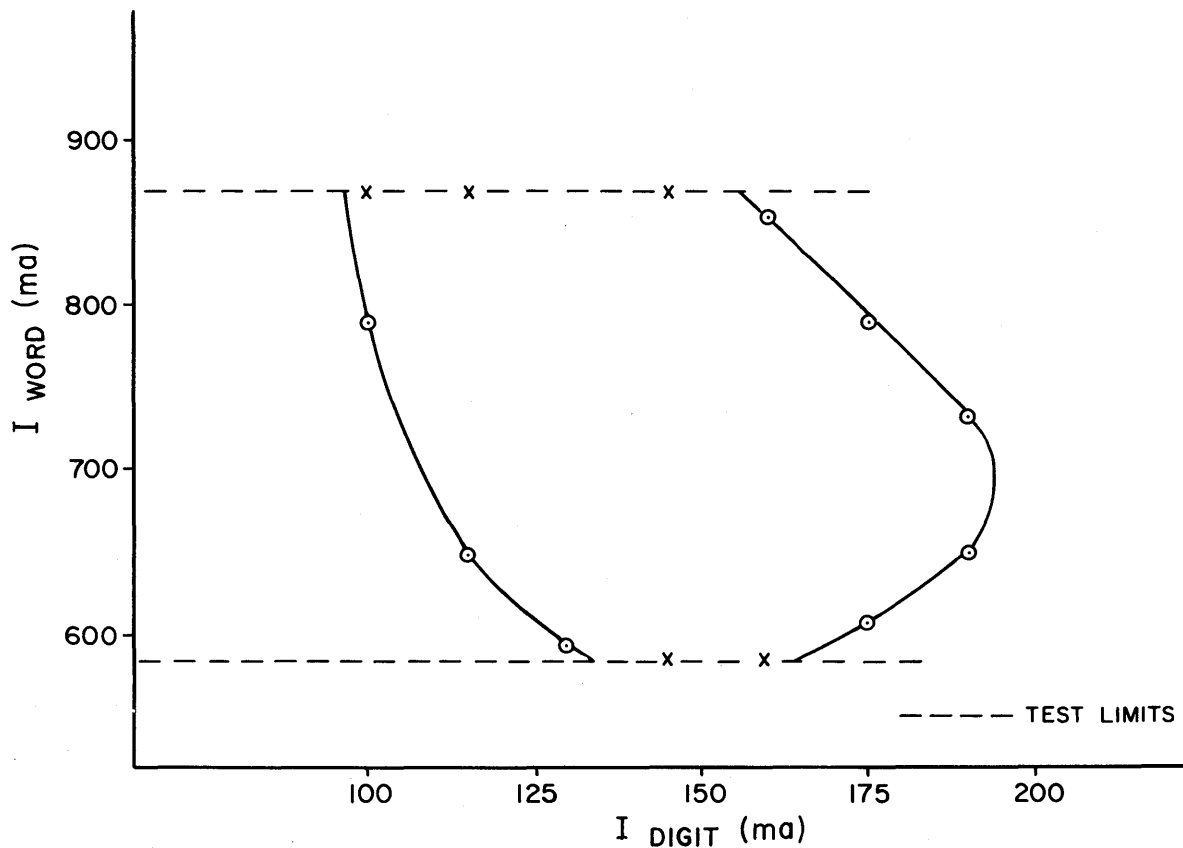


Figure 19. Schmoo plot for a 1024-word stack.

capable of operation at a cycle time of 200 nanoseconds with good margins. The memory is currently in production and several modules will have been evaluated by November 1966.

ACKNOWLEDGEMENTS

Grateful acknowledgement is given to the many engineers and technicians in the memory element design, mechanical design, and circuit groups in the Advanced Memory System Department for the design and development effort that made this memory system possible.

REFERENCES

1. A. V. Pohm, et al, "Large, High Speed DRO Film Memories," *Proceedings of the Intermag Conference*, 1963.
2. W. Dietrich, et al, "The Design of a One-Million Bit 100 Nanosecond Magnetic Film Memory," *ibid.*
3. Q. W. Simkins, "A High-Speed Thin-Film Memory—Its Design and Development," *AFIPS Conference Proceedings* vol. 27, part 1, Supplement (1965).
4. H. W. Katz, et al, "Proposed Low-Frequency Measurement Standard for Magnetic Films," *IEEE Transactions on Magnetics*, vol. MAG-1, no. 3, p. 218.

A ROTATIONALLY SWITCHED ROD MEMORY WITH A 100-NANOSECOND CYCLE TIME

Bruce A. Kaufman, Paul B. Ellinger and H. J. Kuno

*The National Cash Register Company, Electronics Division
Hawthorne, California*

INTRODUCTION

Thin film memory techniques are beginning to offer an attractive alternative to magnetic core memories, particularly for high-speed operation. The memory reported in this paper utilizes a plated-wire (Rod) memory device operating in a 512-word 36 bit per word memory system. The DRO mode is employed and operation at a 100-nanosecond read-write cycle time is achieved. In order to attain this high speed, modified concepts of memory circuitry have evolved affecting every aspect of the memory design. Integrated circuit techniques have been employed for all logic and sensing functions and may be utilized to break some of the economically imposed limitations on contemporary memory circuit design. The potential for low-cost batch fabrication of monolithic and hybrid circuits requires utilization of these techniques to secure cost and performance improvements. Accordingly, the design for this memory makes maximum use of the existing and projected capabilities of integrated circuits.

MEMORY DEVICE CHARACTERISTICS

The storage element used in this memory is an anisotropic thin film of permalloy plated on a beryllium copper wire 10 mils in diameter¹ (see Fig. 1). During the plating process, a direct current flows in

the substrate wire, producing a magnetic film with a circumferential easy axis and the capability of rotational switching. This geometry offers the advantage of a closed flux path in the quiescent state and makes it possible to use relatively thick (10,000 Å) films with minimized demagnetization effects, resulting in increased switching output.²

For the read operation, the magnetization of the film is rotated out of the circumferential direction by application of an axial field orthogonal to the easy axis. The resulting flux change induces a voltage in the substrate wire which is a unique indication of the

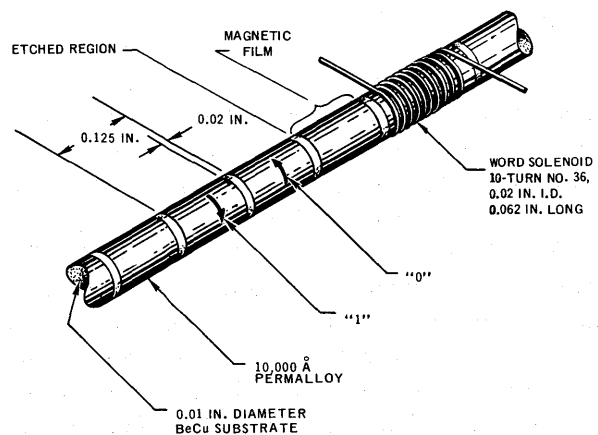


Figure 1. Memory device.

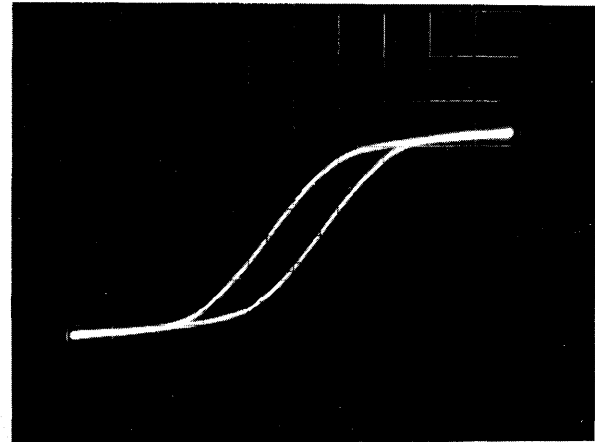
state of magnetization. In order to reduce sensitivity to skew and dispersion, a high overdrive in the word direction is used. This drives the film beyond the H_k value and demagnetizes the bit during read, giving rise to DRO operation.

Wall motion, although a second-order effect, can cause creep. Low-level bit-to-bit interaction along the Rod is also possible, and is caused by the application of a disturbing digit field in the presence of small repeated transverse fields such as fringing from an adjacent bit position. Typically, this problem has been severe on rotationally switched memories.^{3, 4}

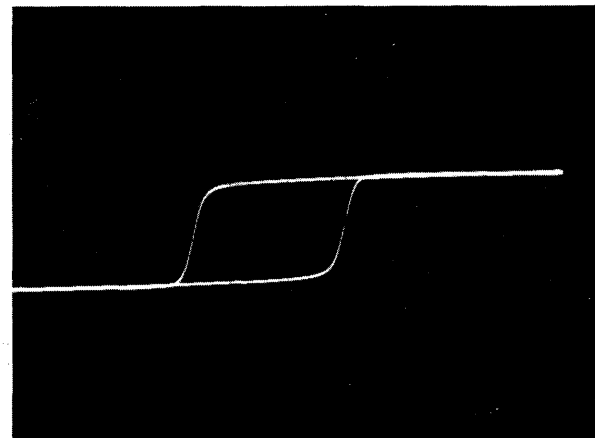
One approach to eliminate the interbit interaction consists of controlling the anisotropy field and the wall motion threshold. In the present state-of-the-art of plated-wire element manufacture, this approach appears to be rather difficult to achieve; therefore, a more direct approach of on-line etching of the plated wire to provide physically isolated magnetic regions is used. In this process, the plated wire is selectively covered with a chemical resist, following continuous plating with permalloy. The exposed area is etched off through an on-line process, thus yielding isolated magnetic areas (Fig. 1). This procedure limits the creep to the physical limit of the bit region and has played an important part in reducing adjacent bit creep. The on-line etching process introduces an additional step in the plating operation; however, this step is easily controllable and has not caused serious yield or processing problems. The plated wire is 80-20 permalloy plated from a modified Wolf's bath, and is approximately 10,000 Å thick. H_k is approximately 5 Oe and H_c approximately 4 Oe. Figure 2 illustrates low-frequency B-H loops of the element.

MEMORY STACK

The commitment to employ a circumferentially oriented, rotationally switched device governs certain aspects of the memory stack organization. To take advantage of the closed flux path, the word current must provide a magnetic field in a direction parallel to the axis of the cylindrical element and the digit field should be applied around the circumference of the element; thus, the digit current must flow down the center of the Rod itself. In addition to the closed flux path, this geometry provides a digit-sense line with the lowest inductance possible, combined with the maximum coupling of the magnetic element to the sense line. Note that no additional winding is



(A) B-H LOOP IN AXIAL DIRECTION
(2 OE PER DIVISION)



(B) B-H LOOP IN CIRCUMFERENTIAL
DIRECTION (5 OE PER DIVISION)

Figure 2. B-H loop. *Top*: In axial direction (2 Oe per division). *Bottom*: In circumferential direction (5 Oe per division).

required on the plated wire, which forms the sense-digit line.

The word field must now be provided by current flowing in either a strip line or a solenoid. As a result of the multiple turns, the solenoid approach yields high field efficiency with a relatively low current level and efficient coupling to the film. The dimensions of the stack are 36 bits by 16 by 32, as diagrammed in Fig. 3.

In order to employ existing tooling, a bit spacing of 0.125 in was used in all three axes. A reduction of this spacing to 0.100 in would double the cubic packing density and can easily be achieved. The word line consists of 36 solenoids in series; the far end is returned in a single pass under the solenoids to pro-

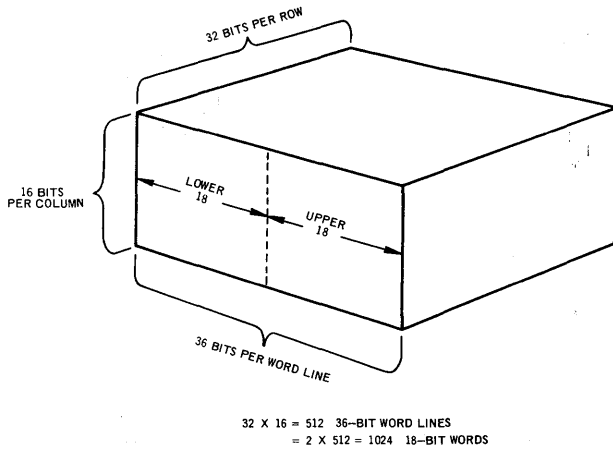


Figure 3. Stack factoring.

vide a low inductance return. This line has a transmission delay of approximately 6 nsec which must be considered in the design of the word drivers. A sense-digit line consists of 16 plated wires, each 32 bits long. The stack is assembled so that the solenoids are axially aligned; the wires are then inserted and are connected in a transposed noise-canceling pattern. For optimum symmetry, the sense amplifier bridges the sense-digit line in the middle,⁵ and the digit current is driven symmetrically at either end, as shown in Fig. 4. The Rods are connected to form the sense-digit line by means of hand-soldered jumper wire; these are formed of light flexible wire to minimize stress.⁶

The sense-digit line is terminated with a single resistor at each end. The digit line is the plated wire with word solenoids distributed along the length at 1/8-in intervals. The solenoids, with discrete cylinders of magnetic plating as a core, form inductances; this inductance with the capacitance between the Rod and the word solenoid forms a lumped constant transmission line. The capacitance at one bit intersection between Rod and solenoid is 0.3 pf. The word solenoid itself with Rods inserted represents 80 nh and without the Rod 30 nh per bit. The Rod itself represents 6 nh per bit. Since the inductive and capacitive coupling between the two halves of the sense line is small compared to the digit-line to word-line capacitance, a suitable representation for this line results if each half of the sense-digit line is considered to form a separate lumped constant transmission line with the word line as the return. Therefore, the only meaningful termination at the ends of the sense-digit lines is that to ground, as shown in Fig. 4.

The sense amplifier then detects the difference in voltage between the two lines. Resistors across the two halves of the sense-digit line at the end serve only to attenuate signals and are not necessary. Because of the three-dimensional mesh characteristics of the actual coupling, the situation is much more complicated than the preceding description. In practice, the sense line termination was empirically determined.

The drive requirements for a 10-turn solenoid are 200-ma word current, with 20-nsec rise and fall times, and 60-nsec width. This produces an axial field of 14 Oe at the center of the solenoid, and 7 Oe at the ends. The digit current is ± 25 ma * with 15-nsec transition time in the nonreturn to zero mode. The digit field is approximately 0.5 Oe at the surface of the film. The digit driver, however, must drive ± 50 ma because it drives two lines in parallel (see Fig.

* "+" and "-" are determined by the information to be written in.

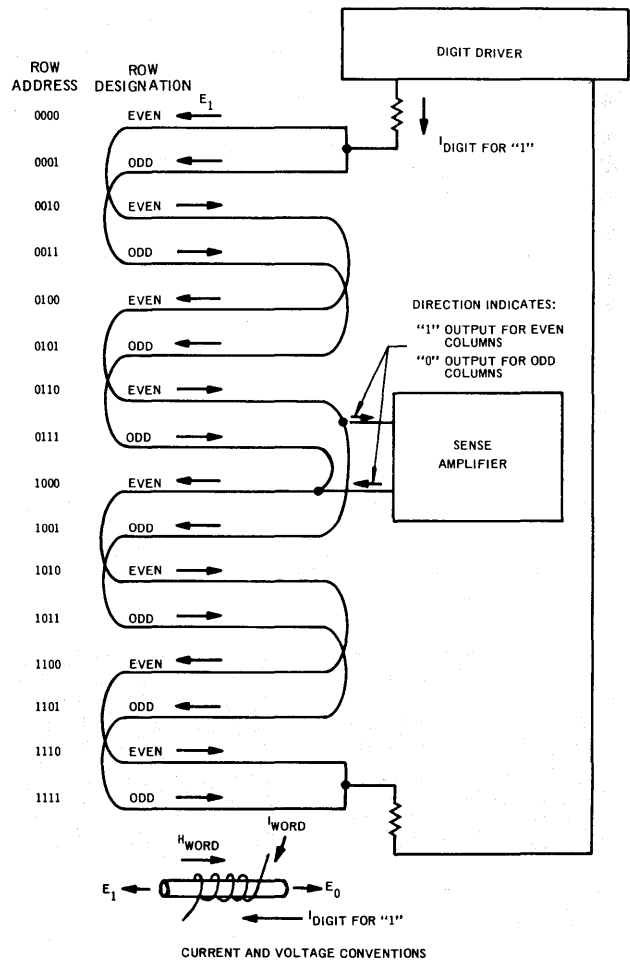


Figure 4. Sense digit-line geometry.

4). Under these conditions, the bipolar sense signal is 50 mv minimum, measured in a standard test fixture. The worst-case signal is ± 20 mv measured at the sense amplifier input terminals. The loss is primarily due to attenuation in the sense line. The time relationship of these signals is shown in Fig. 5. Figure 6 is a schmoo diagram.

Since there is no threshold in the transverse direction, rotationally switched DRO memories must be organized as a linear select matrix.* The ideal approach would be to terminate the line in its characteristic impedance at the far end, and drive from a voltage source equal to the amount of voltage necessary to drive 200 ma into the characteristic impedance. In this case, the dissipation required of each termination resistor would be such that a 6-w resistor would be required for each line. This is prohibitive, primarily because of the required packing density with presently available resistors.

To employ currently available components, the technique described by Pohm⁷ was chosen. This technique requires a termination at the near end of

* A related organization similar to $2\frac{1}{2}D$ selection is possible with rotational switching if the word drive does not exceed the NDRO threshold. In this manner, it is possible to drive a long word line (many data words) and rewrite only new information.

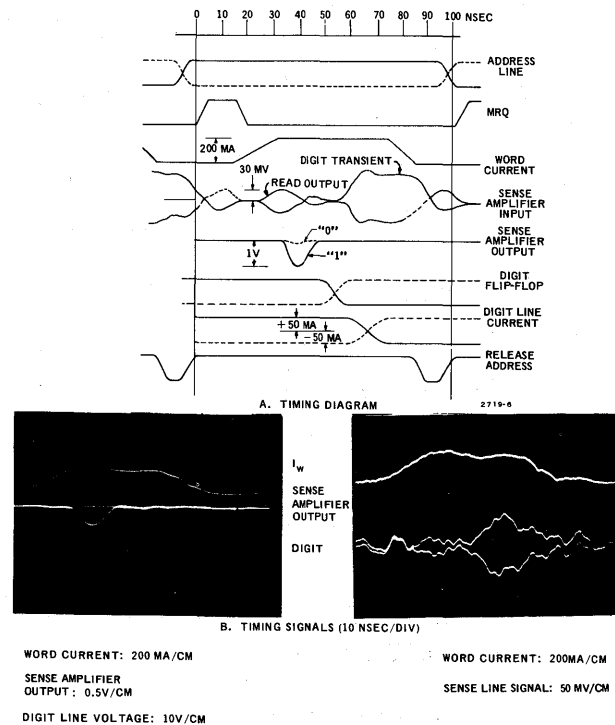


Figure 5. Timing and signals for 10-Mc memory.

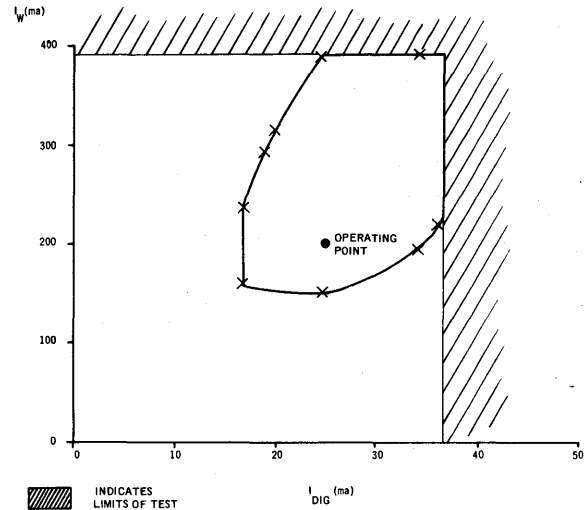


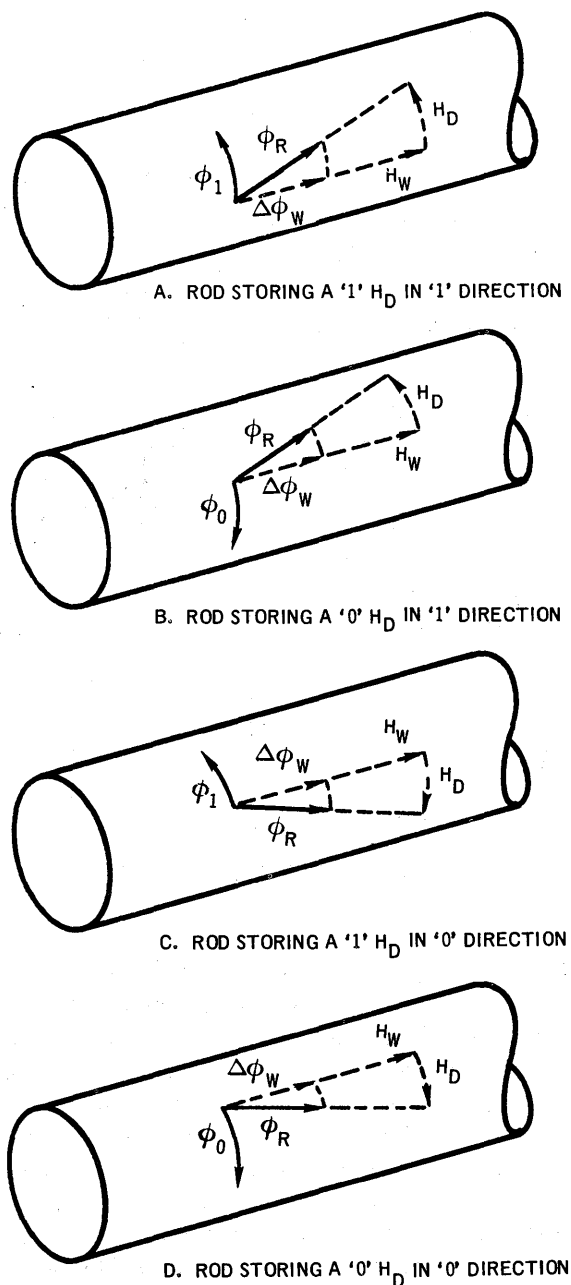
Figure 6. Limits of memory operation at 12 Mc.

the line and a short at the far end. The line must be driven from a current source, so that the first voltage reflection is inverted at the shorted end of the line; in two times the one-way delay (in this case, 12 nsec), the first voltage reflection returns to the sending end. The sending end is terminated so a second reflection does not occur. If the rise time of the driving current pulse is greater than two times the one-way delay, "stair-stepping" of current will be prevented. Furthermore, the power dissipated in the termination resistor is low, and in this case a $\frac{1}{4}$ -w resistor is suitable.

Another significant advantage in using rotational switching is that, to a first order, the inductance of the word line is not a function of the information stored, i.e., the line inductance is not pattern-sensitive. Each time the word current is applied, the same amount of back emf is produced for each word line regardless of information stored. Therefore, current regulation of the word driver is easier to achieve than in a conventional core or Rod memory in which back emf may vary with the data pattern (see Fig. 7). This feature is especially important in a memory where the film device can switch much faster than the rise time of the drive current, because in that case, the shape of the rise of the drive pulse determines the output.

WORD SELECTION SCHEME

Most previous linearly selected memories have used row and column drivers with diodes to prevent sneak paths and isolate the lines. While diodes provide excellent DC isolation, their capacitance causes



- H_W = WORD FIELD
 H_D = DIGIT FIELD
 ϕ_1 = FLUX IN "1" DIRECTION
 ϕ_0 = FLUX IN "0" DIRECTION
 ϕ_R = RESULTANT FLUX
 $\Delta\phi_W$ = FLUX CHANGE IN WORD DIRECTION

Figure 7. Resultant fields in Rod elements under various conditions.

very poor transient isolation, thus making it necessary to drive nearly the whole memory stack during word transitions. This not only causes asymmetric noise on the sense line, but also complicates the design of the word driver, since this parasitic capacitance must be charged before current can flow.

An alternate technique, using one transformer for each word, has also been proposed.⁸ While this isolates the capacity of unselected lines, this has economic limitations, since high-speed transformers are expensive and difficult to manufacture to tight tolerances. Ferrite memory cores have been used as switch cores in a similar scheme, but at much lower speed. Because of the recirculation delay, the word current must be on for approximately 60 nsec to achieve a 100-nsec cycle time. This would require a duty cycle greater than 50% for the word transformer, and would represent an extremely difficult circuit problem.

An approach originally proposed a number of years ago for linear select memories consisted of using one driver per word. This is a desirable technique which eliminates sneak paths and reduces the parasitic capacitance to be driven by each word driver. When transistor costs were relatively high, this approach was judged economically prohibitive; however, because of integrated circuit technology, this technique is worthy of consideration. High quality transistors, suitable for word driving in discrete form, are only slightly more expensive than the two-line diodes presently specified for most high speed core memories. Improved performance justifies the use of one transistor per line. Additional savings result if silicon chips consisting of single or multiple devices are attached to a ceramic substrate for improved heat transfer and higher packing density. Cost estimates have indicated that these chips could be available for approximately \$.10 each in large quantities.

In order to eliminate recovery problems associated with a transformer, it became necessary to use a direct-coupled circuit. The circuit as used is shown in Fig. 8. The selection scheme is shown in Fig. 9.

This circuit solves the problem of driving the line with a constant current while the line voltage is changing. When current is first driven into the line, the short at the far end is not yet effective and the line exhibits its characteristic impedance. Thus, the voltage across the line will rise to $I_0 Z_0 / 2$, (25v) as shown in Fig. 10. Later, the short becomes effective

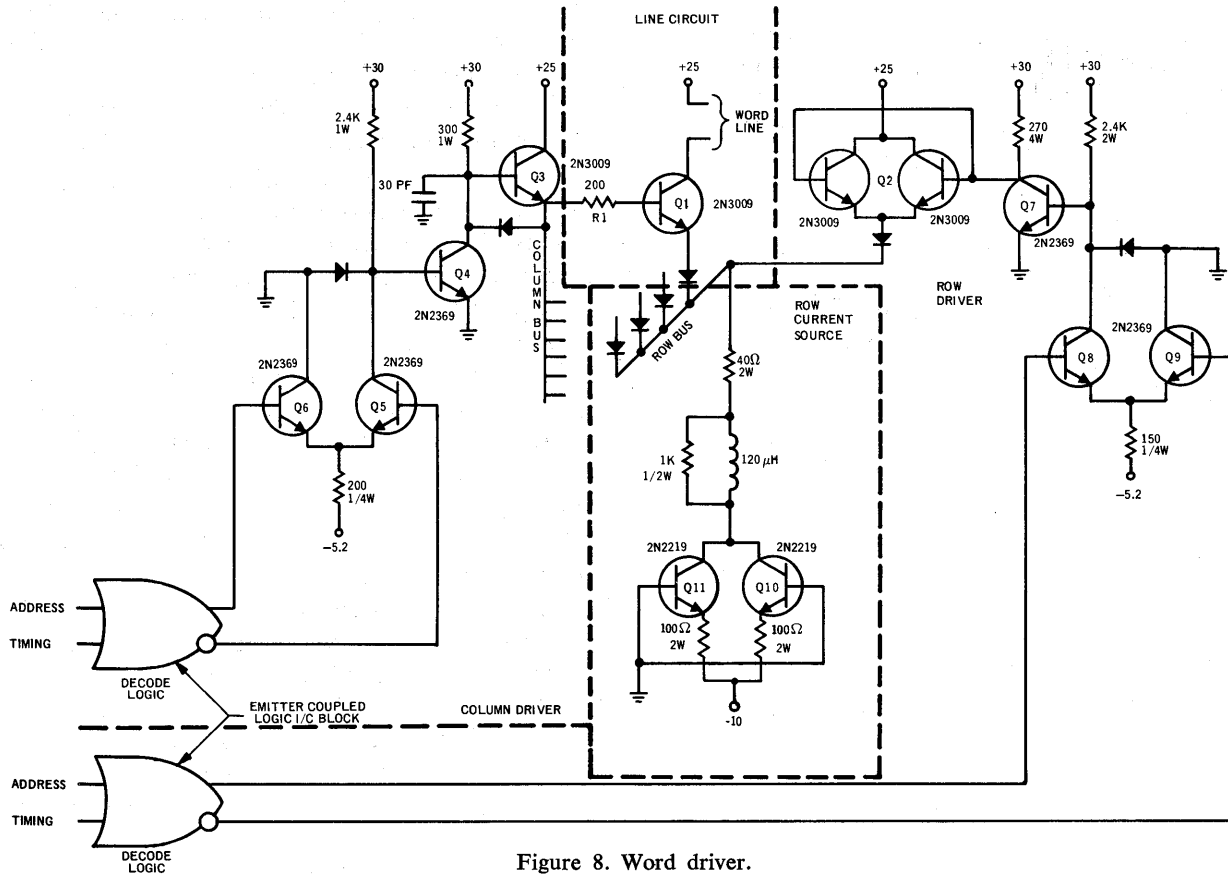


Figure 8. Word driver.

and the voltage across the line becomes only $I_q r_l$, or very nearly zero.

The word driver circuit has a transistor for each word line with one end of the word line connected to a DC voltage (+ 25v); the sending end of the word line, which is terminated, is connected to the collector of the line transistor. This provides better isolation than if the line were connected to the emitter, since there are no other connections to the collector terminal.

In the selection scheme, a row of line transistors, Q1, have their emitters connected to a constant current source with a dummy transistor (alternate current path), Q2. All word lines are quiescently at + 25v. When the dummy is turned off, the constant current source provides whatever voltage is necessary to maintain constant current. A negative voltage is developed on the row bus and if one of the line transistors is saturated, this voltage will be the 25v ($I_q Z_o/2$) plus the transistor drop (approximately 0.5v). The column driver, Q3, causes one line transi-

tor to saturate by applying a slowly rising voltage to the bases of all line transistors in its column.

The signal delivered to the base of the line transistor is shaped so that just enough current to keep the line transistor in saturation is drawn at any time. This is essential since the base current flows through the emitter and into the current source, thus subtracting from the line current. A current-limiting resistor, R1, is also used to insure that the current is evenly distributed to all bases. An emitter-follower, Q3, is used to drive the high capacitance of the line transistor column bases. Unselected line transistors have 25v reverse bias on their base emitter diodes. Since this exceeds the rating of most transistors, a diode was added in series with the emitter of each line transistor. This line driver configuration requires one transistor, one diode, and one resistor for each word line. In a production version of this memory, integrated circuits would be used; the cost of an entire row of these line circuits would be about the same as that of an integrated circuit diode array of similar size.

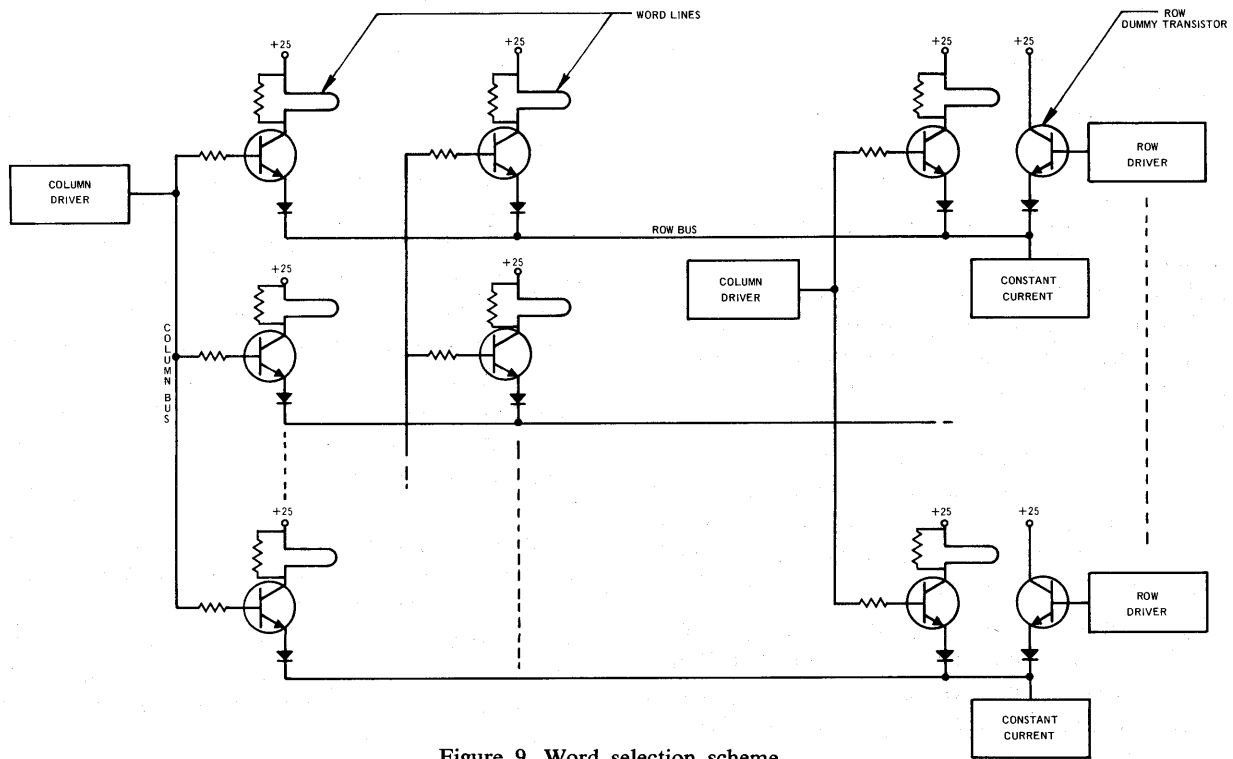


Figure 9. Word selection scheme.

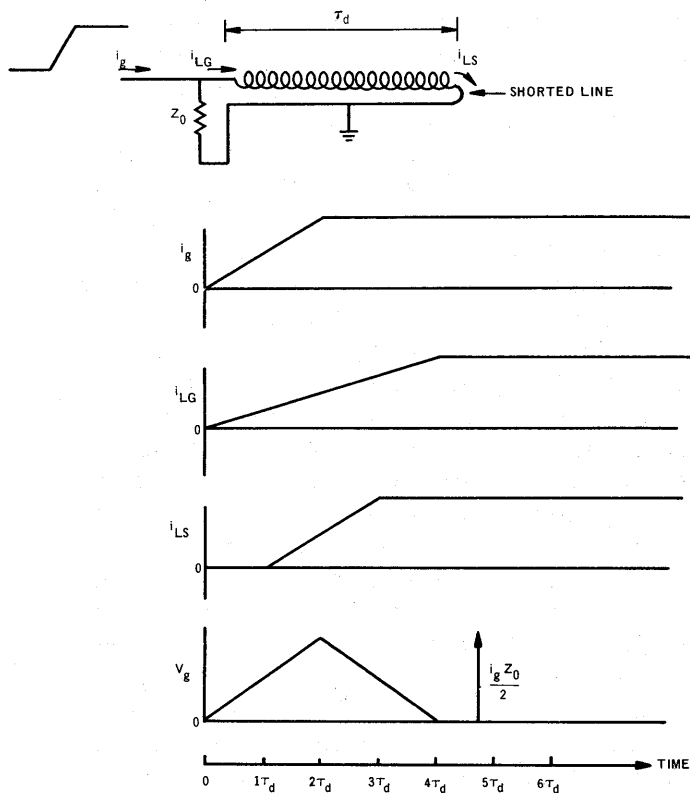


Figure 10. Word driver waveforms.

Design of the constant current source for this word driver presented another difficult problem. Recovery problems at this speed and duty cycle obviate the use of a conventional inductive current source. Because high voltage compliance at high speeds is necessary to prevent pattern shifts, transistors of sufficient speed and power rating to form the current source, and operable in the common-base mode, would be ideal. Since transistors of this type are still costly, a combination of the two methods in series was used. The high-speed compliance is provided by Q4 and Q5 in parallel. Care must be taken in selecting this transistor type, as well as the layout to minimize capacitance at the collector node. This will cause overshoot or ringing and loss of compliance (dynamic range). Recently, high-speed, high-power transistors utilizing multiple emitter designs have become available so that a true transistor constant current source would now be possible.

The word selection scheme, using one transistor per line, retains the row-column decoding feature by connecting the bases of the line transistors together in columns and the emitters together in rows. This has the additional advantage of isolation, so that a larger memory can be segmented and driven from

multiple drivers for each row or column. In a conventional diode selection scheme this isolation does not exist, so there is an upper limit to the size of memory that can be operated for any given speed for a given factoring.

DIGIT DRIVER

One of the most severe problems of a high-speed memory is recovery from the digit transition noise. Classical memory organizations utilize a pulse for the digit drive with some resulting circuit advantages (particularly the use of low-duty cycle transformer coupling); however, this causes two digit transitions per cycle.

The digit driver for this memory must provide 25 ma per side or 50 ma total current at approximately 10v. Although discrete circuits are used in the present design, properly designed monolithic circuits could be employed for this function.

The use of a nonreturn-to-zero (NRZ) digit drive⁹ is a unique feature, and requires only one digit transition per cycle in the worst case. This approach alleviates several difficult recovery problems of the sensing system.

The NRZ mode requires that the digit current flow at all times, even during the read cycle. The digit drive is reversed only if necessary to write the desired information. With the high word drive levels used, only a slight theoretical difference in the output signal occurs, depending on the direction of the digit drive.

An additional advantage of the NRZ digit circuit consists of lower worst-case power dissipation than for an RZ circuit, since the worst-case PRF of the digit driver is equal to one-half the memory cycle rate. For example, at a 10-Mc rate, the digit driver would only cycle at a 5-Mc rate, and in many accesses, it would not cycle at all. This reduces the average dissipation which is increased by transitions.

An additional circuit requirement is that flip-flop or storage action must be provided within the digit driver circuit. The digit driver circuit utilizes two totem-pole drivers with current-determining resistors providing drive directly to both ends of the digit line. These totem-pole drivers are formed from a PNP-NPN pair, and have the load connected to the emitters to gain speed; thus, the totem-pole driver is actually a complementary pair of emitter-followers. The storage is provided by two emitter-coupled logic gates, cross-coupled in a latch circuit. The schematic

of the digit driver-sense amplifier recirculation loop is shown in Fig. 11.

SENSE AMPLIFIER

Perhaps the most difficult circuit design in a computer memory is the sense amplifier. Various requirements such as high speed, fast recovery time, DC coupling, high gain, gain stability, reduced size, and low cost, appear to conflict with one another. A DC coupled long-tail-pair approach has been used to eliminate pattern-shift problems resulting from overload closely followed by small input signals.¹⁰ This approach offers a practical solution to the sense amplifier requirements. Modifications to adapt the techniques proposed in that paper for much higher speed operation are discussed in the following paragraphs.

At very high speeds, the word noise problem is such that common mode rejection at high speeds is extremely important, since good stack design converts capacitively coupled word noise into a common mode signal on the sense line. High common mode rejection is difficult to obtain at high speeds due to capacitive unbalance between two halves of a long-tail-pair. A "balun" transformer provides better common-mode rejection at nanosecond speeds than is possible with an amplifier, and alleviates DC tolerance problems associated with connecting the amplifier directly to the digit line. The hazards in controlling the amplifier bias levels with direct coupling to the sense line have been described in a previous report.¹⁰

Two chips on a common header have been used to provide DC stability resulting from similar thermal environments.¹⁰ This technique has been used in the present sense amplifier, in the form of an emitter-coupled logic gate as an input stage.¹¹ The configuration provided by Motorola Type MC 356 may be used as a differential amplifier with slight modifications. One approach would be the use of two I/C emitter-coupled logic stages in cascade; this can be accomplished at lower speeds, but being designed as a logic device, the dynamic range is limited. Therefore, to achieve a higher-gain bandwidth amplifier with improved dynamic range, the second stage uses two chips in a common header in a long-tail-pair followed by emitter followers for power driving, and diode clipping to prevent saturation of the following logic stage. A strobe is applied at the emitters of the second stage during the digit transient, thereby enhancing recovery. The first stage has sufficient dy-

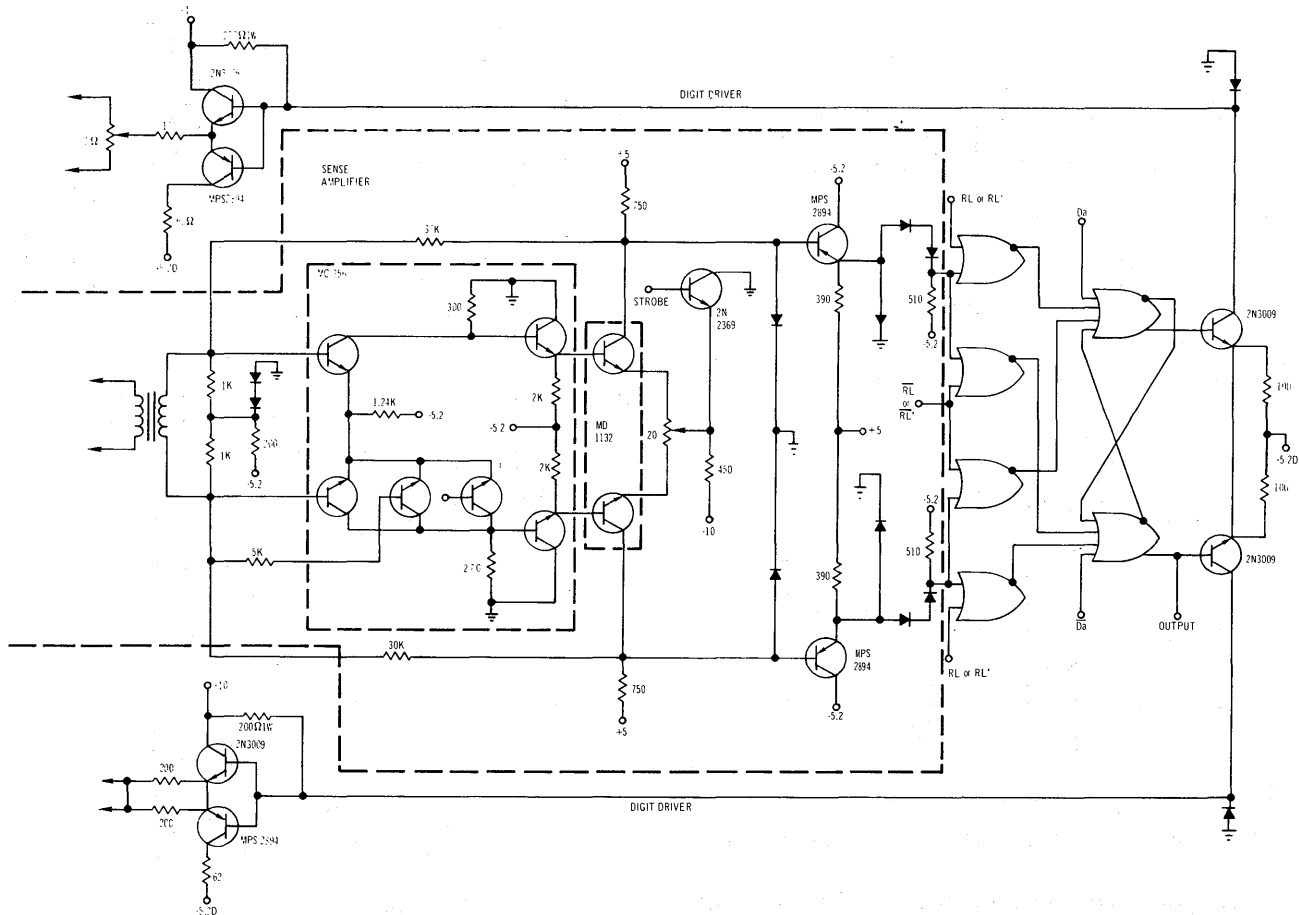


Figure 11. Sense amplifier and digit driver.

dynamic range to handle the input, as well as the digit transition, without overload.

A fast emitter-coupled logic gate is used as a threshold element which has approximately a 300-mv "gray zone," that is, a zone of uncertainty. A DC transfer characteristic for this gate is shown in Fig. 12 and the applied signal is as indicated.

The tunnel diode provides an ideal threshold element because of temperature and current stability and uniformity from one tunnel diode to the next. Tunnel diodes are expensive in comparison with present day transistors, and do not seem amenable to monolithic integrated form. The main disadvantage of the tunnel diode discriminator, however, is the low-voltage output and the necessity for amplification to logic levels after the sense of the output has been determined. This requires additional circuitry and delay time, thus increasing the recirculation time of the memory.

The use of a high-speed emitter-coupled gate provides a threshold element compatible with all logic

levels, thereby eliminating level-shifting or gain-changing with resultant delay and tolerance problems. Such a gate has a propagation time in the order of 2.5 nsec and requires no resetting, as does the tunnel diode discriminator. A temperature-stable threshold results, and perhaps more important, the gate tracks with the sense amplifier because both can be operated from common supply voltages.

The more common approach to sense amplifier in high-speed memories, DC cascaded long-tail-pairs, requires excellent DC stability over the operating temperature range. This stability can be achieved only with common thermal environment for both halves of the sense amplifier. To eliminate complex adjustments, both halves of the amplifier should be as similar as possible, especially with respect to temperature coefficients. This similarity is a natural consequence of building the entire amplifier on a single chip.¹² Monolithic circuitry provides this unique advantage and offers better performance than the dis-

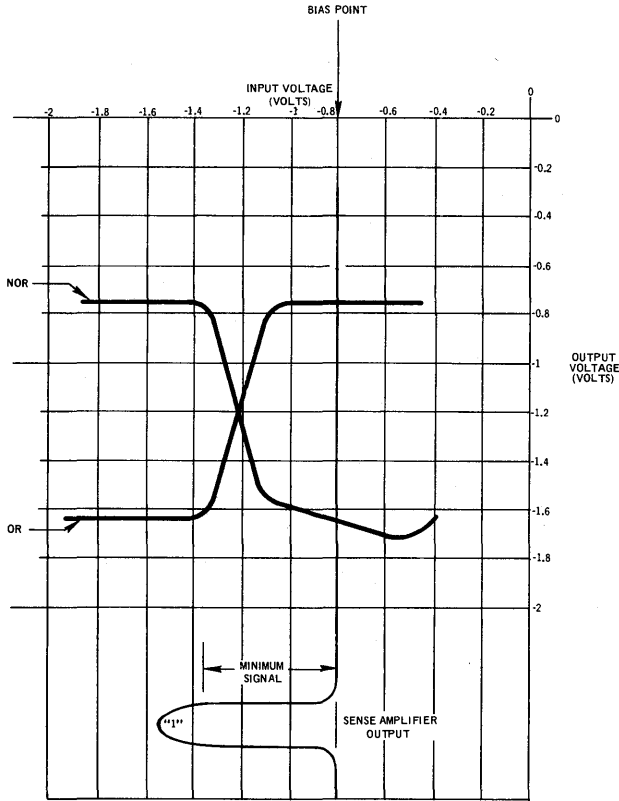


Figure 12. MECL threshold element transfer characteristic.

crete or hybrid separate-chip approach. The inherent simplicity of monolithic sense amplifiers leads to superior performance and eventually, lower cost than possible with discrete equivalents.

ADDITIONAL CIRCUITRY

As mentioned previously, the memory design was implemented with fast (2.5-nsec propagation time) emitter-coupled gates for all logic, address decoding, and timing functions. The basic ECL circuit is shown in Fig. 13. The emitter-followers included in these blocks allow driving of matched 100-ohm lines. These serve as delay lines for timing, as well as distribution lines for delivering various timing and logic pulses throughout the system.

Figure 14 shows the major system components. The only areas where the emitter-coupled logic is not used are the word and digit current drivers; these areas are interfaced to the ECL by a "power" emitter-coupled logic to convert the 800-mv logic swing to that suitable for driving high-level circuits.

The basic circuit used to convert the 800-mv signals to higher levels is shown in Fig. 15. This cir-

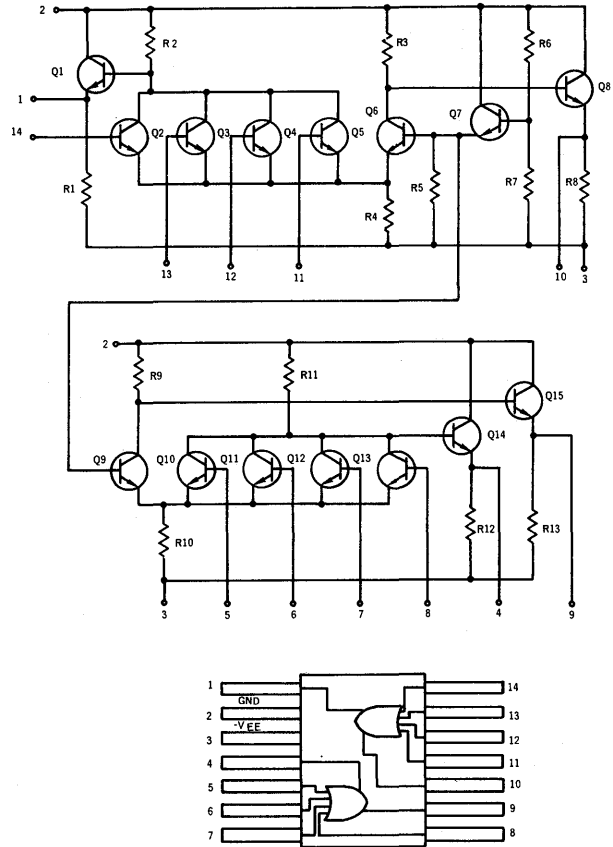


Figure 13. High-speed emitter coupled logic circuit.

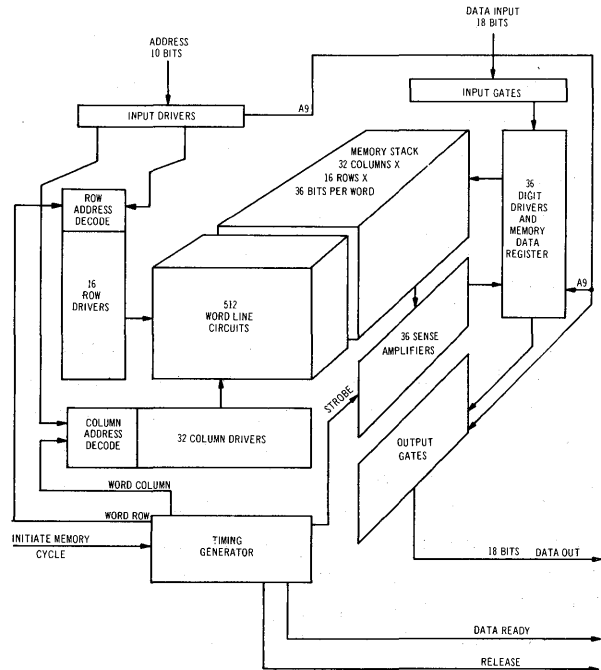


Figure 14. 10-Mc memory block diagram.

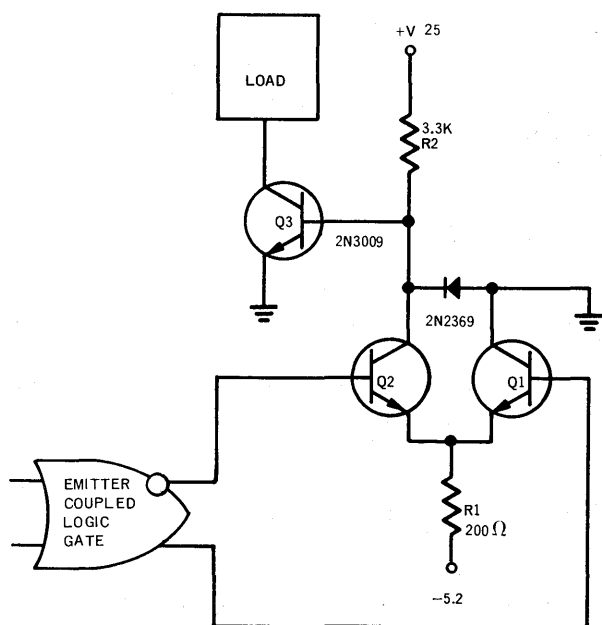


Figure 15. Current mode switching circuit.

cuit employs current-mode switching to obtain fast rise and fall times. Both resistors act as current sources, with the current being switched to alternate paths. When Q2 is conducting, the current from R2 flows through Q2 and into R1. The other possibility occurs when Q1 supplies current to R1, and the current from R2 flows into the base of Q3, causing Q3 to conduct. Fast Q3 turn-off is due to the anti-saturation diode which allows R1 to have a greater current than R2. This excess of current draws the stored charge from the base of Q3, and cuts Q3 off quickly. After Q3 is cut off, the excess current is supplied by the antisaturation diode.

PACKAGING

Considerable planning, both electrical and mechanical went into the package design of the memory. At these speeds, the package design bears an intimate relation to the electrical performance, and care is required to effect a producible design that does not compromise high-speed performance. The packaging of this memory is of unique design, with cards arranged in a cluster around the stack, which is used as a basic frame. This configuration eliminates long leads from the word and digit-sense circuits. This constituted a major problem in other types of memory packaging where these circuits were encased in conventional card cages. Figures 16 and 17 show the packaging arrangement. In addition, the printed circuit boards used are of two-layer construction with

a ground plane in the middle. This layout not only provides an electrostatic shield, but also improves the high-frequency ground system. The cards which plug into the stack also connect into a wiring frame which contains wire-wrapped logic and interface connections over a ground plane.

SUMMARY

In designing this memory, the need for a new approach to memory circuitry was apparent. A direct-coupled word driver matrix drive, and careful attention to stack balance and transmission line characteristics is required. Packaging becomes more important as speeds increase and the memory stack is designed so that circuit cards plug directly into the stack itself. Extensive use was made of ground planes in sandwich circuit cards for shielding and noise reduction.

Monolithic integrated circuits, in addition to their attractive costs, afford opportunities for improved performance. These are utilized in the sense amplifier as a threshold device and as part of the amplifier itself. The emitter-coupled logic gates are used to achieve fast logic speed. The logic level is 800 mv, which made it necessary to design all drivers to operate from this voltage. The timing generator also utilizes ECL gates in conjunction with ordinary

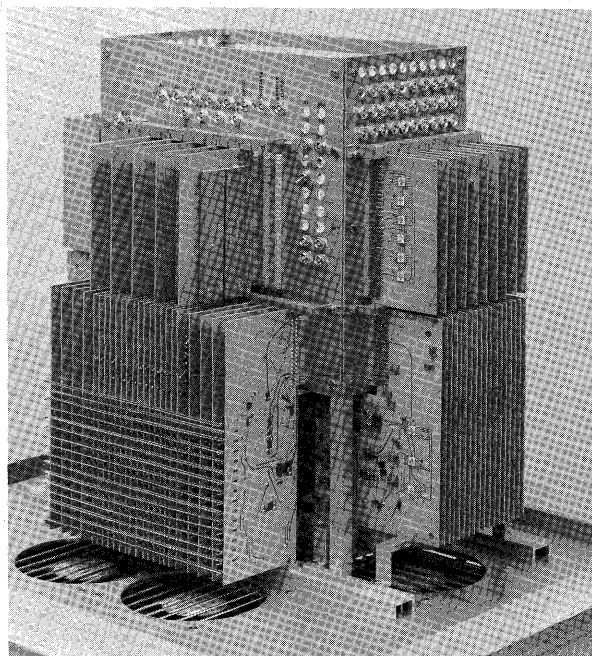


Figure 16. 10-Mc memory (front).

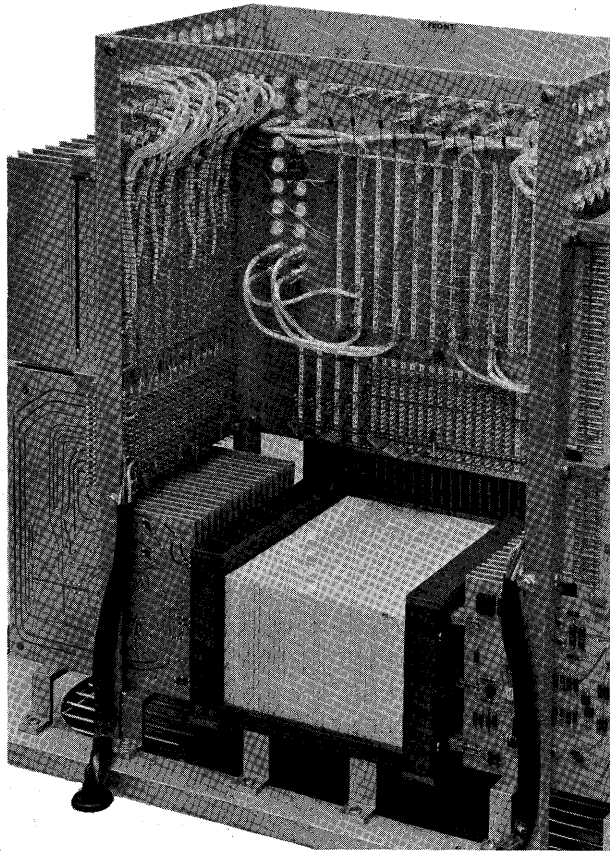


Figure 17. 10-Mc memory (back).

coaxial cable for delays to form the required timing signals.

The rotationally switched memory device has such an inherently fast switching time that the speed of this memory is determined by the stack electrical properties and drive current rise times. High overdrive in the word direction combined with a relatively thick (1 micron) film yields high output signals. This high overdrive makes DRO operation necessary and makes the memory more susceptible to adjacent bit creep, but it tends to minimize the effects of variations in device properties. Creep is limited to a fixed but acceptable range by interbit etching of the film.

ACKNOWLEDGMENTS

The authors are pleased to acknowledge the support and assistance of the Thin Film Memory Device

Project under Dr. H. White in supplying the Rods, the contributions to the sense amplifier design by W. Wong, the packaging design work of R. Sloppy, and the skillful assembly work of Miss M. Roberts, Miss R. Reale, and Mrs. J. Greenwell.

REFERENCES

1. B. Kaufman and E. Ulzurrun, "A New Technique for Using Thin Magnetic Films as a Phase Script Memory Element," *Proc. FJCC*, 1963.
2. H. J. Kuno and P. Ellinger, "Demagnetization Effects in Cylindrical Thin Magnetic Films," *Proc. IEEE*, vol. 54, pp. 991-92 (1966).
3. U. F. Gianola, "Disturb Thresholds in Cylindrical Film Memory Wire," *J. Appl. Phys.*, vol. 34, pp. 1131-32 (1963).
4. A. H. Anderson, T. S. Crowther and J. I. Raffel, "Drive Current Margins for Magnetic Film Memories," *J. Appl. Phys.* vol. 34, pp. 1165-66 (1963).
5. R. Laupheimer, "An Improved Union of Digit-Select Lines and Sense Lines in Word Select Ferrite Core Memory Systems," *Proc. IEEE*, vol. 53, pp. 432-33 (1964).
6. H. J. Kuno, "Effect of a Torsional Stress on a Cylindrical Thin Film Element," *ibid*, pp. 1754-55 (1965).
7. A. V. Pohm, et al, "Large, High Speed, DRO Film Memories," *Proc. Intermag. Conference*, 1963, pp. 9-5-1 to 9-5-14.
8. J. B. Jones, B. J. Steptoe and A. A. Kaposi, "The Design of a 4096-Word One-Microsecond Magnetic Film Store," *Proc. Wescon*, 1962.
9. "Project Lightning, Third Phase, Third Quarterly Progress Preport," Remington Rand-Univac (Dec. 1, 1960-Feb. 28, 1961), vol. I (AD 263 110), pp. 63-65; vol. II (AD 263 109), pp. 113-16.
10. B. A. Kaufman and J. S. Hammond, "A High Speed Direct-Coupled Magnetic Memory Sense Amplifier Employing Tunnel Diode Discriminators," *IEEE Trans. on Electronic Computers*, EC-12, pp. 282-95 (1963).
11. S. T. Robertson, "Build a Differential Amplifier from Logic Gates," *Electronic Design*, Apr. 26, 1965.
12. D. R. Breuer, "Integrated High-Frequency D. C. Amplifier," *Wescon Paper 2.2*, 1964.

A 500-NANOSECOND MAIN COMPUTER MEMORY UTILIZING PLATED-WIRE ELEMENTS

James P. McCallister and Carlos F. Chong

*UNIVAC Division of Sperry Rand Corporation
Philadelphia, Pennsylvania*

INTRODUCTION

From its earliest use in digital data processing systems, plated wire has shown great potential as a practically ideal element for high-speed, random-access computer memories.¹ Some of the advantages of the plated-wire element are as follows:

1. It can be manufactured in a continuous process.
2. It can be tested in a continuous process.
3. It has a very rapid switching time, about 80 nanoseconds, when driven with a current pulse having a 40-nanosecond rise time.
4. Its output, when driven with a pulse having a 40-nanosecond rise time, is 5 to 10 millivolts; this is sufficient to permit practical sense-amplifier designs and good signal-to-noise ratios.
5. The word current is 800 milliamperes, and the bit current is 40 milliamperes; these are reasonable values for good system design. Furthermore, the back voltage on the drive lines is very small, so that the power dissipation of the driving circuits can be kept low.

6. A practical nondestructive readout (NDRO) mode using identical read and write word currents can be used with plated wires, thereby permitting faster cycle times and economical organization.
7. Plated-wire memories can be inexpensively constructed.

Initial applications of the plated-wire element to memories for aerospace use have successfully demonstrated many of these advantages; notably, the switching time, the output, the low drive currents with their associated low-power drive circuits, and the NDRO mode.^{2,3} Also, work has been reported using the destructive readout mode,^{4,5} and other modes of operation.⁶

This paper describes an engineering model of a 150,000-bit (16,384 words by 9 bits) plated-wire memory. One-half of the maximum capacity was constructed and tested.

REVIEW OF PLATED-WIRE PROPERTIES

The plated wires are made by electroplating an iron-nickel alloy onto a beryllium-copper wire which has first been electroplated with copper. The wire is 0.005 inch (0.13 millimeter) in diameter, and the iron-nickel alloy is approximately 10,000 Angstroms

thick. The wire is plated in the presence of a circumferential magnetic field which is generated by passing current through the wire. The result is an anisotropic magnetic structure that has an easy axis in the circumferential direction.

To make use of the magnetic coating, a combination of circumferential and axial fields is used. The circumferential field is produced by a current through the plated wire; the axial field is produced by current through a loop surrounding this wire (in practice, many plated wires are surrounded by one loop). The loop which carries the word current is referred to as a word line, word strap, or solenoid. Each intersection of a plated wire and a word strap is a storage cell for an information bit. Figure 1 is a simple representation of a plated-wire memory bit. The magnetization of the film can rest in either the clockwise or counterclockwise sense of the circumferential easy direction. These two senses represent the binary one and binary zero identities of the information being stored.

In reading, the plated wire serves as its own sense line. The word current establishes an axial magnetic field that causes the magnetization of the film to rotate toward alignment with the axial field. This rotation changes the flux normal to the loop formed by the sense wire and its ground return and causes a voltage to be generated. The voltage generated may be of either polarity, depending on the binary identity of the information stored in the wire. In the NDRO mode, the film magnetization returns to its original orientation when the word current is removed, thus completing the read cycle.

In writing, the plated wire serves as its own bit wire. At the same time that the magnetization vector is partially rotated by a word-current field, a small current is driven through the bit wire. This bit cur-

rent provides a circumferential field which steers the magnetization to the proper sense. Writing is thus a coincident-current operation in which the bit current must be large enough to switch the film under the active word strap but small enough not to switch the films under the inactive word straps. Writing will be successful as long as there is at least a minimum time-overlap between the word and bit currents. The magnitudes of the word and bit currents can be established by using the same principles applied to planar thin films.^{7,8}

Because the individual bits are not physically discrete elements but are actually portions of a continuous magnetic cylinder, there is a tendency for interference between bits. This interference, which is reversible, occurs when information of the same polarity is repeatedly written into one bit cell. The bits under adjacent word straps may be reversed and cause the wrong information to be read out. An effective technique has been devised to prevent this interference. During a single write cycle, first the complement of the information is written, and then the desired information. Thus, in every case, the magnetic history of a particular bit is balanced in ones and zeros, and no more than two write operations of the same polarity can occur consecutively at any one bit location.

LOGICAL ORGANIZATION OF THE MEMORY SYSTEM

With the introduction of $2\frac{1}{2}D$ and other similar core memory systems, various methods of address selection in the bit and sense dimensions have come into use.⁹ The bit-sense matrix used in the memory described herein is a highly efficient and economical method for reducing the number of word lines. Figure 2a shows a 16,384-word memory of conventional word-organized design. Figure 2b shows the modified word-selection system which uses the bit-sense matrix. The matrix consists of 144 switches between the sense amplifiers and bit drivers on the one side and the plated wires on the other. The function of the matrix is to route the desired signals from the plated wires to the sense amplifiers and to route the bit currents from the bit drivers to the desired wires. The same switch serves both routing functions. Each sense-amplifier, bit-driver combination is associated with 16 plated wires, one of which is selected by the bit-sense matrix. The selection of one of the 16 switches is controlled by 4 bits of the address.

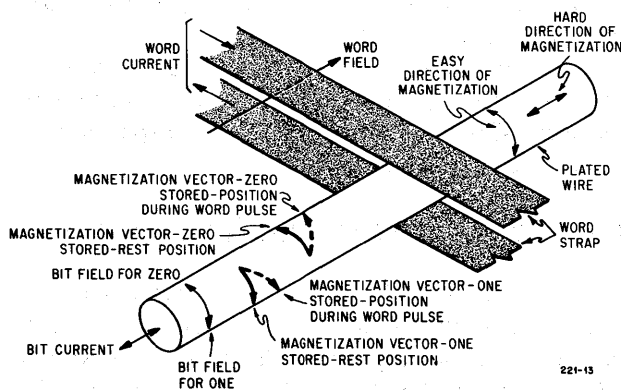


Figure 1. Information storage on plated wire.

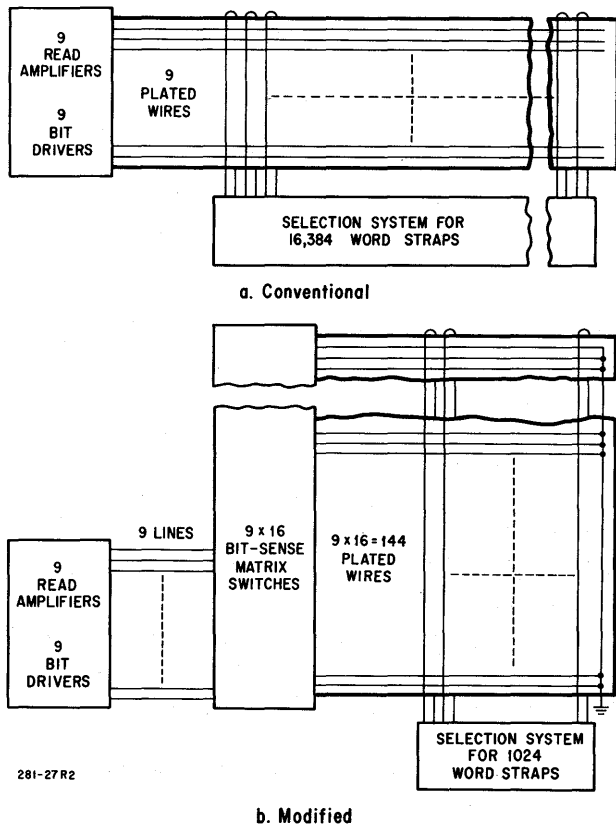


Figure 2. Block diagram of a conventional and a modified word-organized memory.

Thus, what was basically a 16,384-by-9 stack becomes a 1024-by-16-by-9 stack. Also, the length of the bit-sense line in the array is reduced by a factor of 16, thus making large arrays a practicality. The switch accommodates a bipolar bit current of approximately 40 milliamperes and sense signals of a few millivolts without contributing noise. (The switch circuit will be discussed later.) Because of the NDRO mode, the action of the active word strap on the 15 unselected wires per bit channel does not destroy information. The bit-sense circuits which would otherwise be needed for regeneration are not required.

This memory organization is possible only with an element which has NDRO characteristics for both read word current and write word current. Its economy lies in the fact that a single switch element controls selection for both sense and bit drive functions and thus requires an array with a common wire for both functions. A small back voltage on the bit-sense wire during the bit-writing pulse is desirable for sense dimension recovery and for realizing the dual-function switch element.

Figure 3 is a block diagram of all elements of the selection process. The complete memory array is arranged on four planes. Each plane contains 144 plated wires which intersect 256 one-turn word straps; the 144 bits under each word strap comprise 16 words. Each word line is in series with a diode, and all the word lines are arranged electrically into a matrix. The selection lines (called A and B lines) at opposite ends of the matrix are orthogonal to each other. On each plane there are 16 B-selection lines and 16 A-selection lines. Physically, two planes are mounted to a single base plate, one on each side. On each 2-plane assembly, 1 set of 16 A-lines serves both sides, but there is a separate set of 16 B-lines for each side. In the complete array of 4 planes, the 16 A-lines of each 2-plane assembly are driven by separate sets of switches, which have identical inputs. That is, the same A-line is selected on both assemblies simultaneously. Thus, logically, the 1024 word-line diodes are in a 64B-by-16A array, but electrically there are two 32B-by-16A arrays.

The bit-sense matrix consists of 144 W-selection switches which connect 9 wires out of 144 to the 9 sense amplifiers and 9 bit drivers. The W-selection switches are in turn driven by circuits which derive a 1-out-of-16 selection from 4 bits of the address.

OVERALL PACKAGING

Figure 4 is a front view of the complete memory system, and Figure 5 is the rear view. The circuits

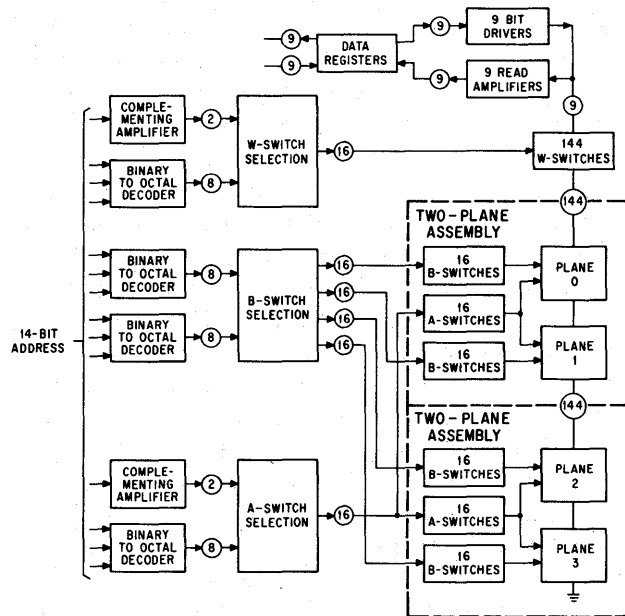


Figure 3. Selection system.

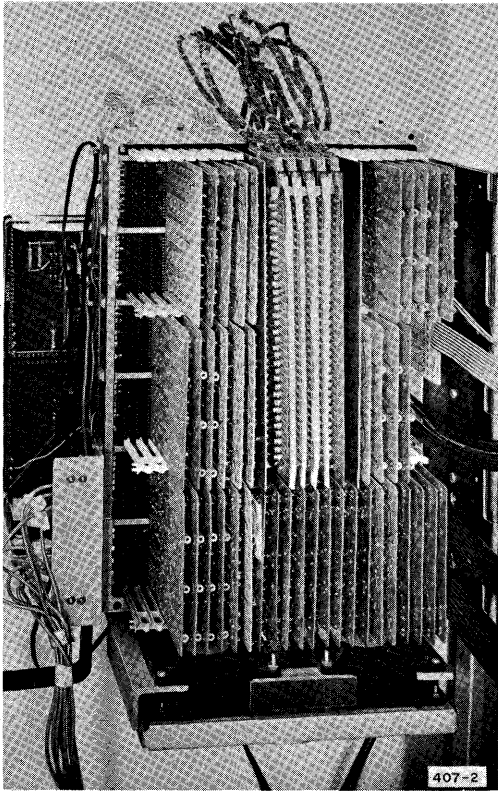


Figure 4. Complete memory system—front view.

are assembled onto 6-inch-by-6-inch (15.3 centimeters by 15.3 centimeters) printed circuit boards. The W-selection switches are assembled on special oversized boards approximately 6 inches by 12 inches (15.3 centimeters by 30.5 centimeters). One such board mounts all W-switches associated with two sense amplifiers. A complete set for one sense amplifier consists of 16 switches for the 1-out-of-16 plated-wire selection plus one switch for a noise-canceling channel. Thus, each board holds 34 switches. These boards contain three copper layers: signal, ground, and selection. Because the dielectric between the signal and ground layers is very thin, noise coupling into the signal paths is held to a low value. The two assemblies containing the four planes are mounted behind the main card library frame. Connectors are mounted directly on each of the two-plane assemblies, which hold the printed circuit boards for A-switch and B-switch selection of word lines and associated circuits. The 144 plated-wire circuits and the 9 noise-canceling wire circuits are connected to the W-switch matrix boards by means of the disconnectable cable assemblies shown at the top of the picture.

MEMORY PLANE AND STACK

The memory plane design successfully meets three different sets of requirements: mechanical, electrical, and magnetic. In the first step of the construction of the plane, oversize pilot wires are sandwiched between two sheets of Teflon*-coated Kapton*, the Kapton being toward the outside. Heat and pressure are applied, and the Teflon flows, conforming to the wires. Next, the ground sheet, the spacer, the bottom word-strap layer, the wire sandwich, and the top word-strap layer are laminated onto a unified assembly, as shown in Fig. 6. The pilot wires in the sandwich are then pulled out, leaving tunnels 0.008 inch (0.2 millimeter) in diameter. Since the plated wires used in the memory are 0.005 inch (0.13 millimeter) in diameter, they can be inserted into the tunnels easily with very little force.

The word straps are etched on a substrate of glass epoxy. Separate sheets are used for the top and bottom, and the ends are soldered together to form a

* Registered trademarks of the duPont Company.

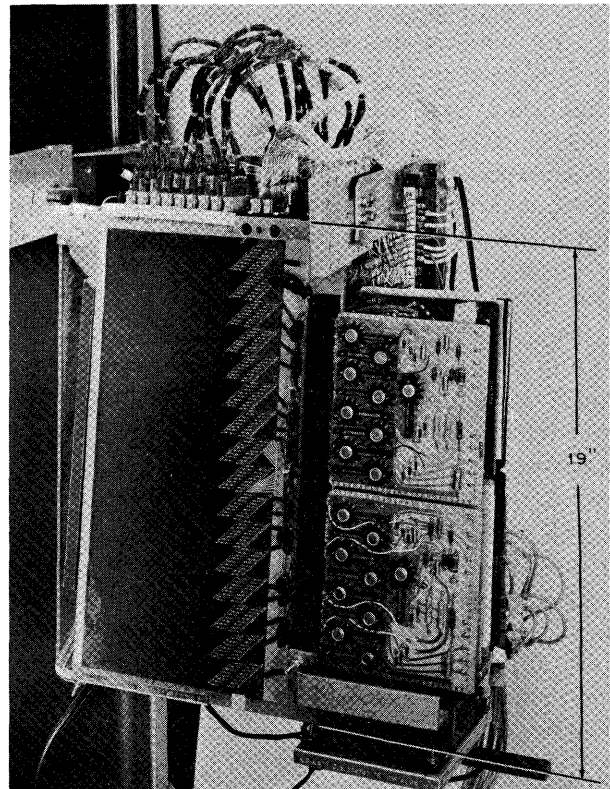


Figure 5. Complete memory system—rear view.

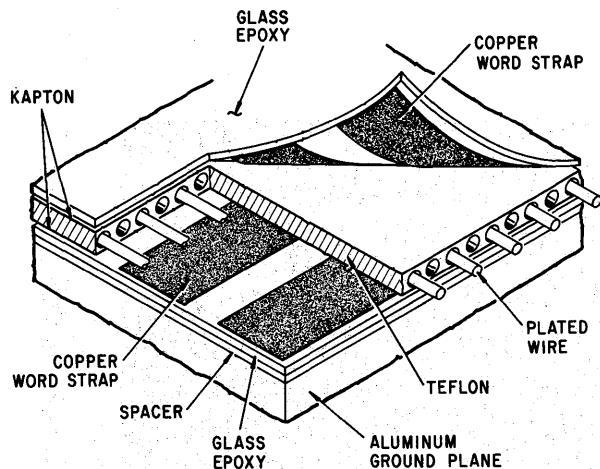


Figure 6. Memory-plane construction.

complete one-turn solenoid. The electrically significant dimensions are given in Table 1:

For every 16 magnetic wires, there is one non-magnetic, noise-canceling wire which serves as the other half of a differential-input pair. In order to avoid discontinuities in the magnetic structure, it is necessary that the spacing of the magnetic wires not be interrupted by the noise-canceling wire. Therefore, the plated wires are inserted into every other tunnel, and the noise-canceling wires are inserted between two plated wires at appropriate intervals. Figure 7 is a closeup photograph of the wire termination area of a plane and shows the plated wires on 0.030-inch (0.76 millimeter) centers soldered to etched copper pads. Figure 8 is an overall photograph of one plane and shows the 256 word-line diodes, the wire ends, and the active plane area.

CIRCUITS

Logic and Control Circuits

The basic logic element chosen is a three-input positive AND inverter (Fig. 9). It is a single-transistor DTL circuit. By adding a diode gate structure to the input, two-level AND/OR logic can be performed. Maximum dissipation is 285 milliwatts, and worst-case delays are 10 nanoseconds for turn-on and 14 nanoseconds for turn-off. Maximum input is 6, and maximum output is $3\frac{1}{2}$ logic loads plus 50 picofarads of capacity.

A 3-transistor, high-power amplifier was designed to drive a maximum of 15 logic loads plus 250 picofarads of capacity. The circuit has a complementary

Table 1. Electrically Significant Memory-Plane Dimensions

	Inches	Milli meters
Word-line spacing, center to center	0.060	1.52
Word-line width	0.040	1.02
Word-line solenoid thickness, inside, face-to-face	0.011	0.28
Word-line conductor thickness	0.0015	0.038
Tunnel inside diameter	0.008	0.20
Plated-wire diameter	0.005	0.13
Tunnel spacing, center-to-center	0.015	0.38
Plated wire spacing, center-to-center	0.030	0.76

transistor emitter-follower output stage (Fig. 10). Worst-case turn-on and turn-off delays are each 10 nanoseconds.

A delay flop (Fig. 11) was designed to generate timing and control pulses. The delay period can be varied from 40 to 250 nanoseconds by adjusting the timing capacitor. The delay flop has the same input and output loading as the inverter.

Word-Selection Circuits

The word-selection circuits select 1 of 1024 word-line solenoids. Figure 12 is a simplified diagram of the word-selection system. A conventional diode matrix is selected by 16 A-switches and 64 B-switches. A current regulator is used to supply a word current of up to 1 ampere with a maximum rise time of 30 nanoseconds. A current sink quiescently accepts the current from the current regulator during standby. After an A-switch and a B-switch are turned on, the current sink is turned off; as a result, the current

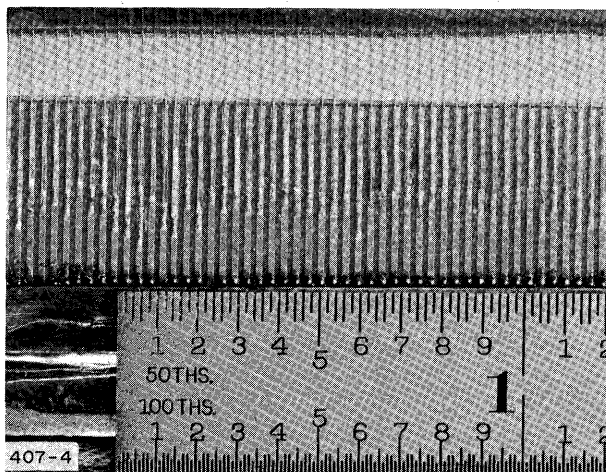


Figure 7. Closeup of wire termination area.

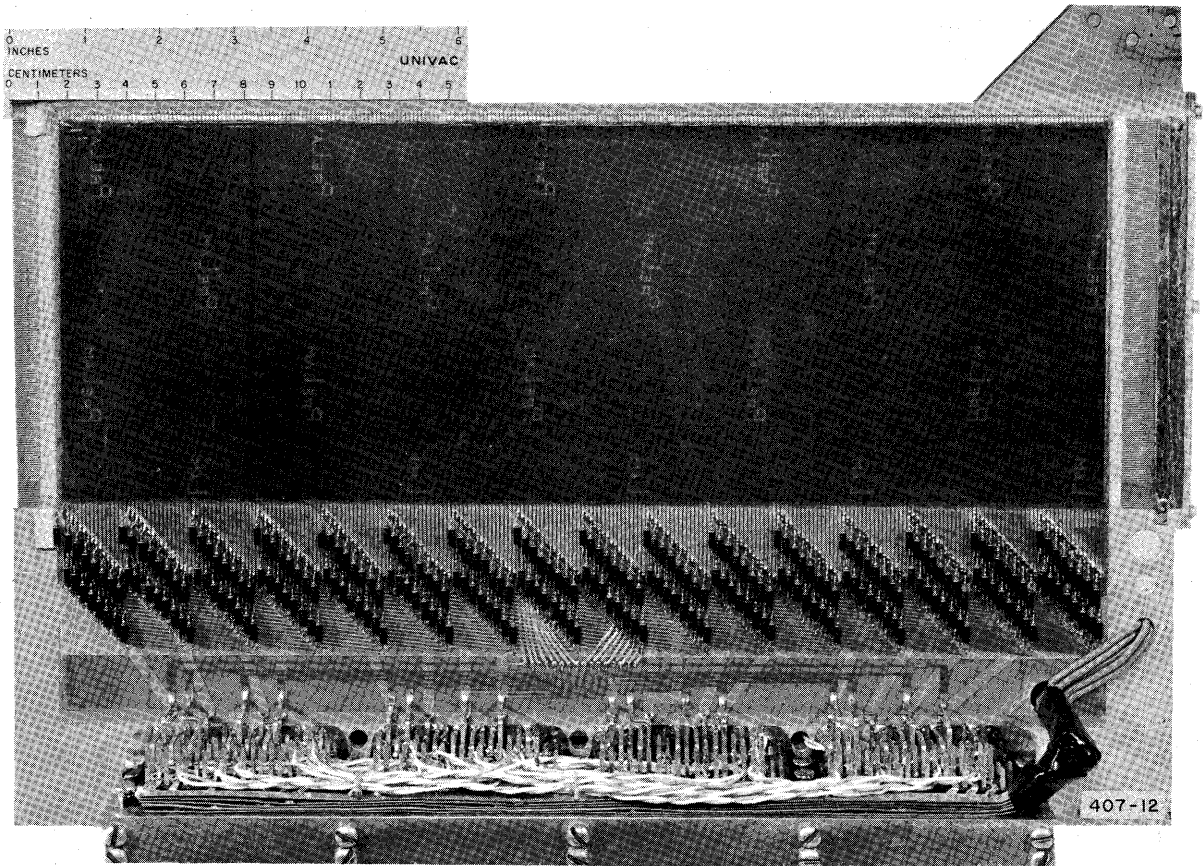


Figure 8. Memory plane.

from the regulator is forced down the selected word line via the selected switches. The B-switch transistors are 2N3015, and the A-switch transistors are 2N3725. The word-line diodes are FD-6's.

Figure 13 is a schematic of the A-switch. Diodes D1, D2, and D3 form an input AND gate for a timing signal and two decoded and amplified address signals. Transistors Q1 and Q2 are emitter-follower

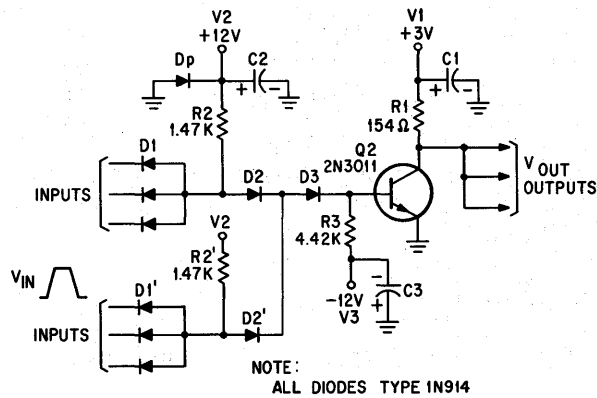


Figure 9. Inverter circuit.

stages with the collector of Q2 driven by the current source.

Figure 14 is a schematic of the B-switch. Six bits of the address are decoded into two groups of eight outputs. One group is amplified and drives the bases of the B-switch transistors. The other group is amplified and drives the emitters of the B-switch transistors.

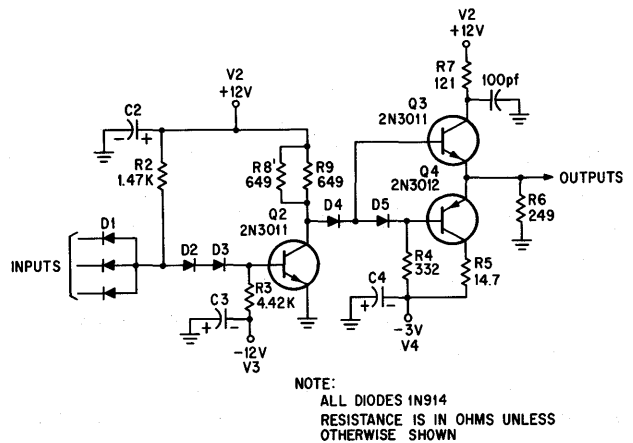


Figure 10. High-power amplifier.

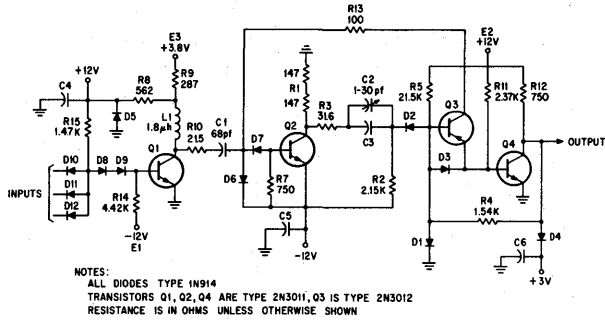


Figure 11. Delay-flop circuit.

sistors. By matrixing in this way, the number of transistors required in the B-switch system is reduced. One other input of the B-switch transistor is a charging circuit. This charger is an emitter-follower circuit that pulls the collector of Q1 back up to 12 volts after the word current ends. One charger serves the whole memory.

W-Matrix Switch (Bit-Sense Matrix)

The W-matrix switches are a set of gates between the plated wires and the read amplifiers and bit drivers. Because of the modified word-selection structure described earlier, 16 words are energized by each word-group line. During a read operation, 9 W-switch elements gate the 9 signals from 1 word selected out of 16 into the read amplifiers. During the write operation, the same elements gate the bipolar bit currents into the desired plated wires. The W-switch matrix element consists of two complementary transistors, as shown in Fig. 15. When a

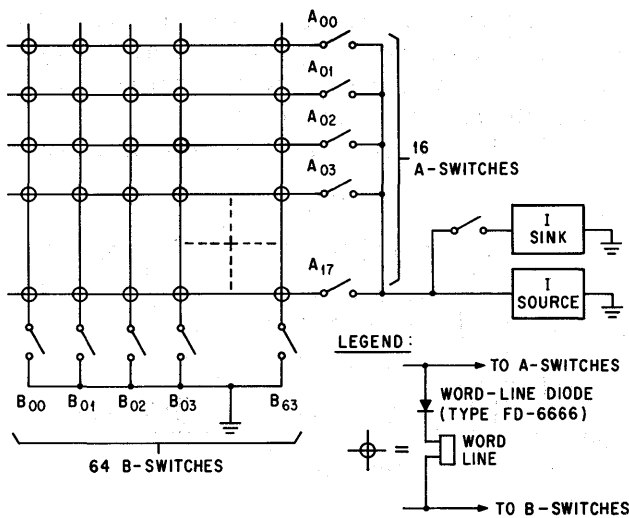


Figure 12. Word-line selection system.

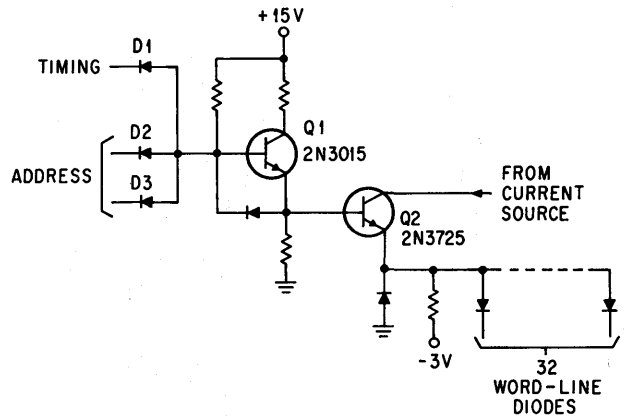


Figure 13. A-switch circuit.

circuit is selected, turn-on currents are applied to the bases of both transistors. The PNP transistor, when thus selected, has a collector-to-emitter drop of 1 to 2 millivolts and a dynamic resistance of 18 ohms. For a write operation, the PNP transistor conducts the positive bit current, and the NPN transistor conducts the negative bit current.

Read Amplifier

There are four different causes of a DC shift, which operates on the signal as it enters the read amplifier. These causes are listed as follows:

1. The W-switch adds a DC offset voltage directly.
2. The average value of the actual signal beyond the switch is not zero, since the wire output is switched into the amplifier only part of the time.
3. The writing process impresses a very large, non-zero-average voltage on the amplifier terminals.

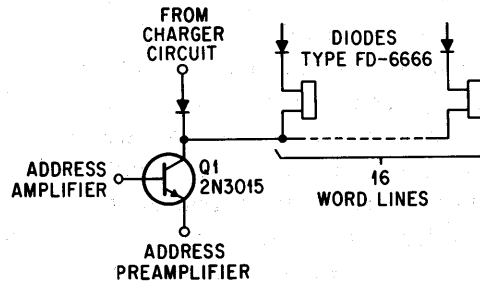


Figure 14. B-switch circuit.

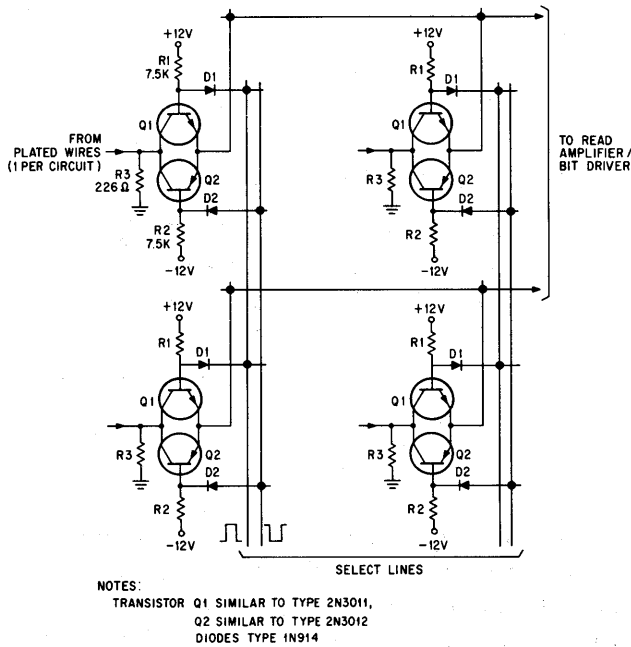


Figure 15. Bit-sense matrix circuit.

4. The sense line itself takes a significant amount of time to recover completely from the bit current.

Several configurations of read-amplifier design were tried in which AC coupling, noise-protection switches, common-mode rejection chokes, and other devices were used. The most successful design, and the one used in this model, employs a DC differential-input amplifier followed by a DC restorer. The DC restorer might more properly be called a DC corrector. Its function is to shift the amplifier output so that just prior to readout the amplifier is at zero reference level regardless of any DC shifts which may actually exist at the input.

Figure 16 is a block diagram of the complete bit dimension circuitry associated with each of the nine bit-channels. It was found convenient to package all elements shown in this diagram on a single-printed circuit board, one board for each bit channel. This made a considerable reduction in backboard wiring, noise pickup, and wiring delays. A single register serves for either input or output information. During write cycles, the contents of the register are gated against two timing signals to generate first the complement writing pulse and then the normal writing pulse, as described earlier. These two timing signals are referred to as Phase 1 and Phase 2, respectively.

During read cycles, the one-zero decision is made by applying the signal and a 20-nanosecond-wide strobe pulse to 2 transistors with a common collector resistor. If both transistors cut off, an output results. Preceding this stage are the DC amplifier and DC restorer, previously described.

Because of the bipolar wire output signal, the strobed detector can be biased very close to a zero-voltage input, so that a one-signal will trip the detector, while a zero-signal will reverse and provide noise protection. Hence, the AC signal-to-noise ratio is quite good and permits the use of high-gain sense amplifiers. A limitation on gain does exist, however, because of the following constraints:

The DC restoration process unavoidably introduces some noise, which must be kept small compared with the signal level at the restorer. This calls for gain ahead of restoration.

DC offsets (or DC noise) at the input must not drive the amplifier out of linearity ahead of the DC restorer. This limits gain ahead of the restorer to less than $\left(\frac{\text{total output range}}{\text{total DC input noise range}} \right)$.

Thus, the signal-to-noise ratio at the detector will be expressed as

$$\frac{\text{signal pulse amplitude} \times \text{gain ahead of restorer}}{\text{noise introduced by DC restorer}}$$

Gain following the restorer cannot improve this ratio.

Figure 17 is a schematic of the amplifier and DC restorer. The amplifier has a differential gain of 52 decibels, with an 8-megahertz cutoff frequency. The common-mode-rejection ratio at 10 megahertz is 46 decibels. The amplifier output continuously charges capacitor C1 through its own low impedance plus that of the restorer transistor, Q8, which is turned on. Just prior to the time a signal is expected, Q8 is

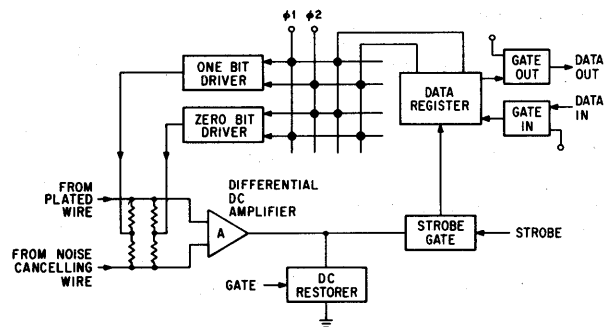


Figure 16. Block diagram of bit dimension circuitry.

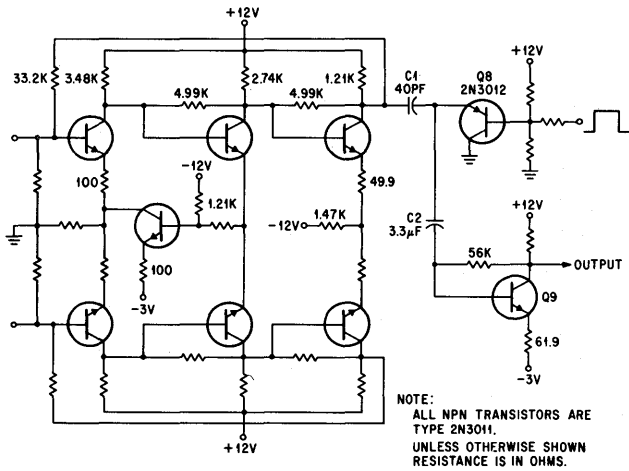


Figure 17. Read amplifier and DC restorer.

turned off, and all variations in output are transmitted to the next stage. While Q8 is on, the time constant of C1 is less than 5 nanoseconds; while Q8 is off, the time constant is 1 microsecond. Capacitor C2 was necessary to isolate the DC conditions of Q9, and as such is a compromise in performance. Variations in waveshape cause DC shifts on this capacitor which are equivalent to a 1-millivolt input signal. In operation, the overall signal-to-noise ratio is typically 5:1.

SYSTEM OPERATION

Figure 18 is a timing diagram of the system functions. The basic objective was to perform noisy operations as quickly as possible, so that the wire output signals could occur at a quiet time. No objectionable noise is introduced by the rise of the word current proper. The most sense-line noise is caused by turning on the B-switch, which moves 16 word-line solenoids through a large voltage excursion, and by turning on the low-level W-matrix switch, where momentary unbalances in base current during turn-on become a direct input to the amplifier. When operated and tested, the memory had a 500-nanosecond cycle time and a 300-nanosecond access time. The limitations on cycle time in this unit are almost entirely in the circuits. In every case techniques are now available to improve upon the demonstrated performance.

TEST AND RESULTS

Figure 19 shows the test pattern sequence which is the worst-case for the memory system described.

The test includes simple operation, nondestructive readout operation in the presence of many disturbs, and operation in the presence of adjacent-bit disturbs. After a normalizing history, every other bit along each wire is written one time. These are called test bits, and those which are skipped are called adjacent bits. The test bits are never again written during a given test. A fourfold cycle of passes through the memory is continued indefinitely: A first pass reads all bits, the second pass writes adjacent bits to be the same as the test bits, the third pass reads all bits, and the fourth pass writes adjacent bits to be opposite from test bits. In this manner, the cycle is continued.

The system operated successfully with this set of patterns for extended periods of time representing many millions of word disturbs and bit disturbs on every test bit. It was also used successfully for several months as a main store for a small computer.

CONCLUSIONS

The construction and operation of the memory system described indicate that such a system is indeed a practicality. It seems a certainty that plated wire memories will become a very important member in the hierarchy of storage systems to be used in the computers of tomorrow.

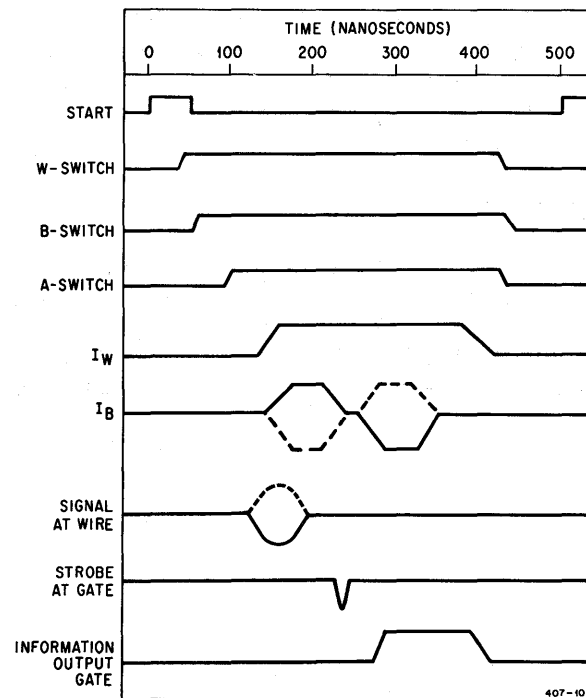


Figure 18. Timing diagram of system functions.

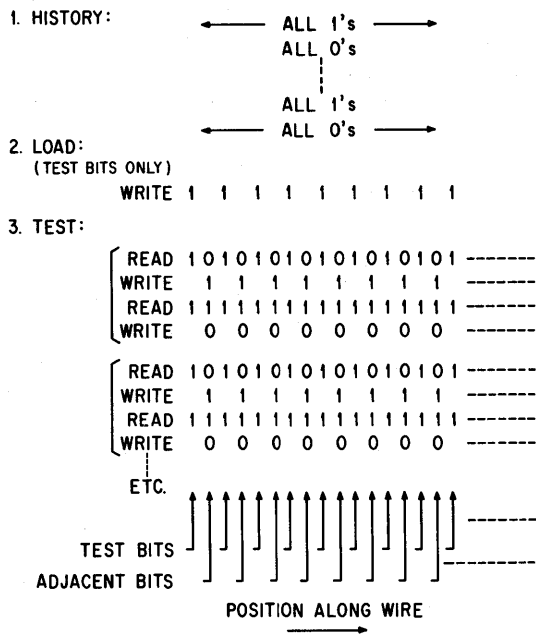


Figure 19. Worst-case test pattern.

ACKNOWLEDGMENTS

The significant contributions to this work are too numerous to list. However, we especially wish to acknowledge the work of Dr. J. S. Mathias, who made the wire; Mr. G. R. Reid, who was responsible for the design and construction of the planes; Mr. P. Zakarian, who did much of the circuit-design work; and Messrs W. J. Bartik and G. A. Fedde, under whose direction the project resulting in this memory was undertaken.

REFERENCES

1. T. R. Long, "Electrodeposited Memory Elements for a Nondestructive Memory," *J. Appl. Physics*, vol. 31, p. 123S (1960).
2. G. A. Fedde, "Design of a 1.5 Million-Bit Plated-Wire Memory," *Proc. Eleventh Annual Conf.*

- on Magnetism and Magnetic Materials, *J. Appl. Phys.*, vol. 37, pp. 1373-75 (1966).
3. —, and G. H. Gutfroff, "A Reliable Very Low Power Plated Wire Spacecraft Memory," *Proc. National Electronics Conference*, vol. 20, pp. 681-86 (1964).
4. H. Maeda and A. Matshushita, "Woven Thin-Film Wire Memories," *Proc. Intermag. Conference*, Apr. 1964, pp. 8-1-1 to 8-1-6.
5. T. R. Finch and S. Waaben, "High-Speed DRO Plated-Wire Memory System," *Proc. Intermag. Conference*, Apr. 1966, paper 12.3.
6. M. Bienhoff, J. Camarata and M. Sherman, "Some Considerations in the Design of Plated Wire Memory Systems," *Proc. IEEE National Symposium on Batch Fabrication*, Apr. 1965, pp. 88-102.
7. A. V. Pohm and E. N. Mitchell, "Magnetic Film Memories, A Survey," *I.R.E. Trans. on Electronic Computers*, EC-9, p. 308 (1960).
8. H. J. Oguey, "Theoretical Hysteresis Loops of Thin Magnetic Films," *Proc. of I.R.E.*, vol. 48, p. 1165 (1960).
9. T. J. Gilligan and P. B. Persons, "High Speed Ferrite 2½D Memory," *Proc. Fall Joint Computer Conf.*, 1965, pp. 1011-21.

BIBLIOGRAPHY

Fedde, G. A., "A Low Power Plated-Wire Memory System," *Sperry Engineering Review*, Fall 1965, pp. 19-22.

Bartik, W. J., C. F. Chong and A. Turczyn, "A 100-Megabit Random Access Plated-Wire Memory," *Proc. Intermag. Conference*, Apr. 1965, pp. 11.5-1 to 11.5-7.

Danylchik, I., A. J. Perneski and M. W. Sagal, "Plated Wire Magnetic Film Memories," *Proc. Intermag. Conference*, Apr. 1964, pp. 5-4-1 to 5-4-6.

Oshima, S., K. Futami and T. Kamibayashi, "The Plated Wire Memory Matrix," *ibid*, pp. 5-1-1 to 5-1-6.

A HIGH-SPEED INTEGRATED CIRCUIT SCRATCHPAD MEMORY

I. Catt, E. C. Garth and D. E. Murray

*Motorola, Inc., Semiconductor Products Division
Phoenix, Arizona*

INTRODUCTION

Computer systems are presently being designed and fabricated using one- to two-nanosecond current-mode logic gates. High-speed scratchpad memories are required in order to utilize this circuit speed effectively.

This paper describes an integrated circuit memory containing 64 words of 8 bits per word, which is compatible in respect to both speed and signal level with high-speed current-mode gates. The memory has a nondestructive read cycle of 17 nanoseconds and a write cycle of 10 nanoseconds without cycle overlap. This is considerably faster than previously reported integrated circuit memories.¹⁻⁴

In addition to high speed, a large degree of system flexibility is achieved by using an integrated storage flip-flop of the type described in this paper. Multiple, independently addressed read channels can be included in the memory allowing simultaneous access to more than one storage location. Separate write address decoding allows a write operation to take place at the same time that one or more read operations are occurring. Possible applications of simultaneous read and write operations will be described later, along with certain constraints.

A description of the device, circuit, package, and system design required to implement a memory of

the type referred to above is contained in the following sections.

SYSTEM ORGANIZATION

A block diagram of the memory system is shown in Fig. 1. As can be seen, the memory has a dual read capability, i.e., any two storage locations can be addressed and read simultaneously. The address decoding is performed in two levels of logic to limit the fan-out to eight and the fan-in to three.

The memory is designed for a particular application and operates in what is referred to as a read-decision-write cycle. This means that following a read operation, a decision is made whether or not to write predetermined data into a predetermined address.

Figure 2 contains a timing diagram for the memory system. In a read-decision-write cycle, an A-Channel and/or B-Channel read address, a write address, and write data are presented to the memory prior to the receipt of a read initiate A or read initiate B pulse. The read initiate pulse starts the memory cycle by clocking the read address into the read address register. The address is decoded and the data contained in the selected word gated onto the output lines where it is then OR'ed to the output data register. The appropriate delayed read initiate pulse clocks the information into the output data

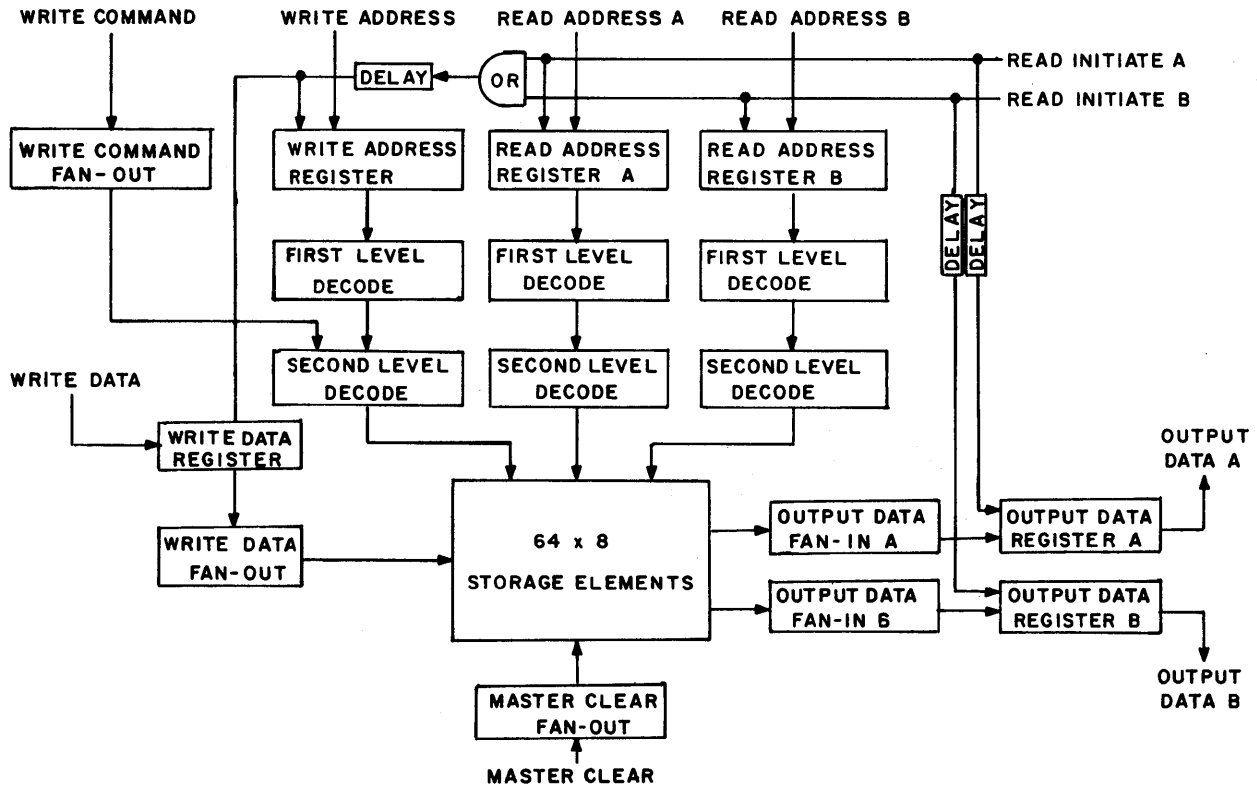


Figure 1. Block diagram of memory system.

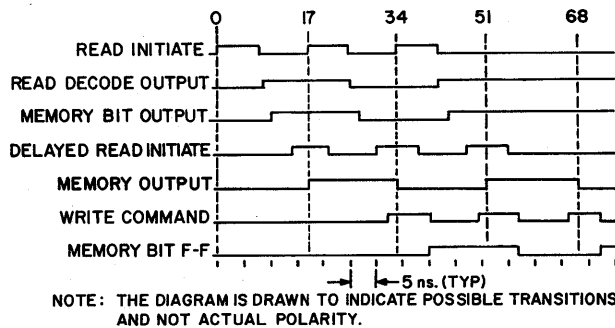
register. The information is then sent to the computer which takes 15 nanoseconds to make the decision of whether or not to write. If the computer decides to write, a write command pulse is supplied to the memory. The write command combines with the write address, which is clocked into the write address register at the appropriate time by a delayed read initiate pulse, and allows the write operation to proceed. The write data is also clocked into the write data register by the same delayed read initiate pulse.

As the timing diagram indicates, the written information is not available to be read from the memory until the third cycle. In the particular appli-

cation for which the memory is designed, it is desirable to have access to the written information on the second cycle. This is accomplished by providing circuitry external to the memory which bypasses the memory whenever a read and write operation occur in the same storage location. The external circuitry contains registers which store the read and write addresses and the new data being written into the memory. When the read and write addresses match during a cycle when a write command is supplied to the memory, the memory output is inhibited and the new data is gated out of the external register and used in place of the memory output. With the external bypass route, the memory performs a full read-decision-write cycle in 17 nanoseconds, the read cycle time.

The block diagram of Fig. 1 indicates a master clear capability. The master clear circuitry can clear the entire memory to ZEROS in 10 nanoseconds. This feature is provided only for the particular application, and it is not required to clear a word to ZEROS prior to writing, since writing is a jam transfer operation.

The output data fan-in block of Fig. 1 is the 64-input OR required in each of the eight bit posi-



NOTE: THE DIAGRAM IS DRAWN TO INDICATE POSSIBLE TRANSITIONS AND NOT ACTUAL POLARITY.

Figure 2. Memory timing diagram.

tions to connect the output from all 64 words to the single output data register flip-flop. The method of implementing this OR is discussed later.

The write command fan-out and write data fan-out blocks are simply extra stages of logic to limit the required fan-out on any node to eight.

MEMORY CIRCUITS

Memory Bit Circuit

Two memory bit circuits such as those shown in Fig. 3 are interconnected on each integrated circuit die to form one bit of two different words. All inputs and outputs are directly compatible with current-mode logic gates. Each bit consists of a gated flip-flop which drives two current-mode read gates.

One of the gated flip-flops is made up of transistors $T_1, T_2, T_3, T_4, T_5, T_6, T_8, T_9$. Transistors

T_8 and T_9 form a current-mode gate which operates one diode drop below normal logic levels. The base of transistor T_9 is connected to the bias voltage, V_{BB} , through the diode formed by transistor T_{10} , and the base of transistor T_8 is connected to the word write input, WW_1 , through the emitter-follower level shift formed by transistor T_7 . In the quiescent state, WW_1 is low and transistor T_9 conducts the current down through transistor T_2 or T_5 depending on the state of the flip-flop. During a write operation, the WW_1 input is pulsed high and the current is switched into transistor T_8 . This causes either transistor T_1 or T_6 to conduct depending on whether the bit or bit input is high. A ONE is written when the bit input is high and a ZERO when the bit input is high. At the end of the WW_1 pulse, the current is switched back into transistor T_9 and the flip-flop is latched

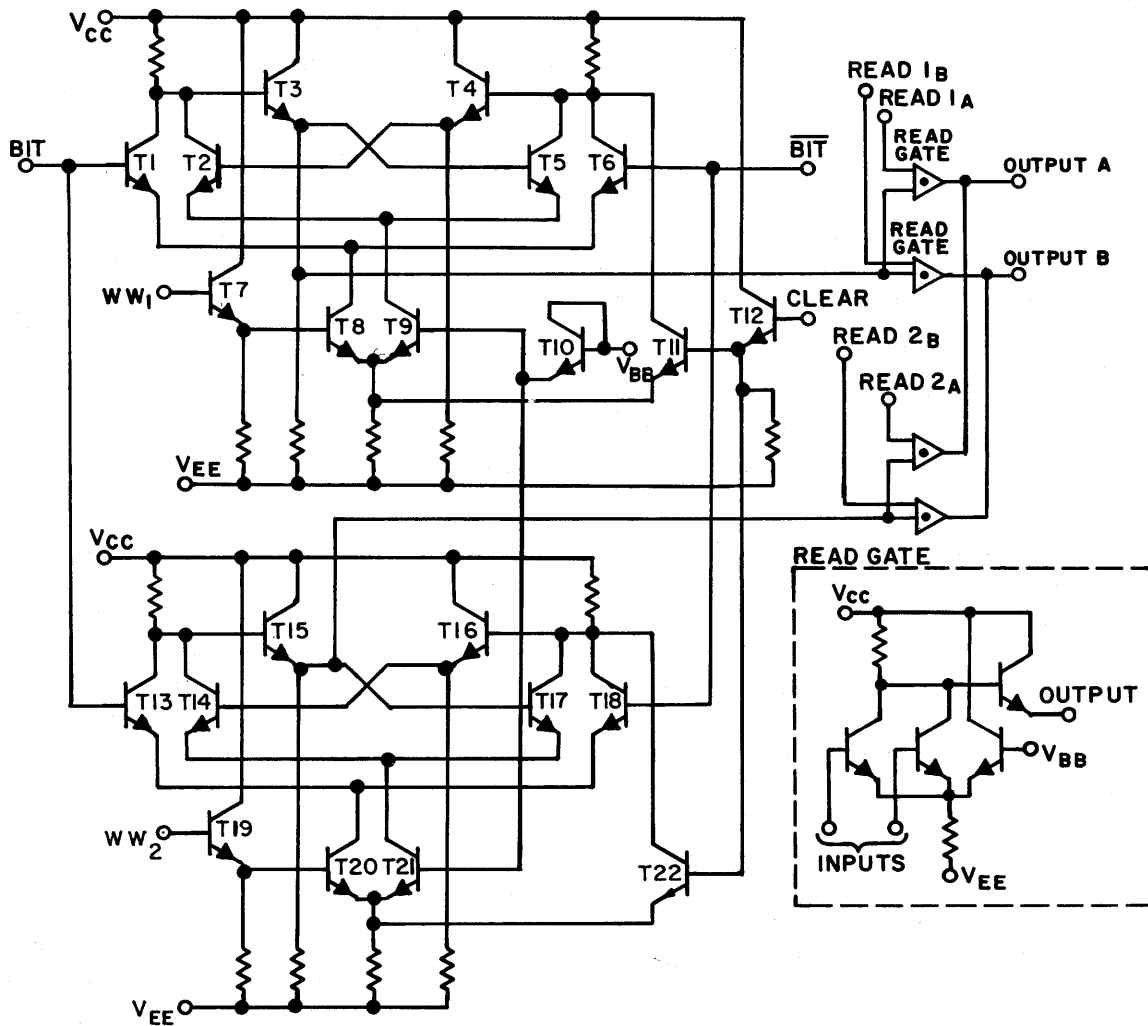


Figure 3. Schematic of memory bit circuit.

in the desired state. The storage flip-flop and the two AND functions required to write selectively into any bit location are thus achieved at the expense of only one unit of circuit power. The delay from the word write input to the slowest output of the flip-flop is approximately 1.7 nanoseconds resulting in a power-speed product of 100 picojoules for the entire function.

It should be mentioned that the bit input can be connected to V_{BB} , thus reducing the number of inputs by one. A ONE is then written when the bit input is high and a ZERO when the bit input is low. The use of single-ended inputs reduces the complexity of the printed circuit board layouts considerably. This is not done in the present system because data taken on a discrete component breadboard indicates that one side of the flip-flop is significantly slower when single-ended inputs are used. This effect is largely due to stray capacitance in the breadboard, however, and the monolithic circuits show very little difference in switching times between single-ended and complementary inputs. Future systems will be constructed using single-ended inputs.

Transistors T_{11} and T_{12} are used to implement the clear function. When the clear input is pulsed high, the current is switched from transistor T_9 into transistor T_{11} thus forcing the flip-flop into the ZERO state. It should be noted that if the collector of transistor T_{11} were connected to another set of bit and bit transistors, a dual write function would be achieved. The delay from the clear input to the flip-flop output is the same as for the write delay, approximately 1.7 nanoseconds.

The read gates are current-mode gates which are capable of driving a 50-ohm load with a delay of 1.5 nanoseconds. Corresponding outputs from the two bits are connected together on the die to produce a tied-emitter OR. Eight such groups are then fed into an 8-input OR gate to achieve the desired 64-input OR. The 8-input gate drives the output data register.

Figure 4 shows a photomicrograph of an integrated circuit die containing two complete memory bits. Double layer metallization is used to provide the interconnections on the die. The die is 50 by 55 mils in size.

A single die is placed in a standard 14-lead flatpack and all leads are used. The total power dissipated in the memory bit package is approximately 275 milliwatts. The thermal design of the memory is discussed in the packaging section.

Register Circuit

The register circuit is a simple modification of the memory bit circuit, and a variation of the same integrated circuit mask set is used to produce both. Two register circuits like those shown in Fig. 5 are contained on each die. As can be seen, the modifications consist of deleting the read gates, eliminating the clear input, and altering what is now the clock input. The resistor values are also changed to allow the circuit to drive a 50-ohm load. The change in resistor values tends to increase the power dissipation; however, the elimination of the read gates reduces the overall circuit power substantially. The total power per die is approximately 150 milliwatts. As before, one die is placed in each 14-lead flatpack. A photomicrograph of a register circuit die is shown in Fig. 6. Two-layer metallization is again used to perform the desired interconnections.

The modifications to the clock input were made to allow tapping of transmission lines with a minimum of reflections. The flip-flops in any particular register are usually grouped in a cluster. If all the flip-flops are connected to the end of a long transmission line, a large reflection results from the capacitive loading. Figure 7 shows how the emitter-follower level shift transistor can be used to buffer the flip-flops from the line. Only one emitter follower is connected to the line and its emitter is used to drive all of the flip-flops in the register.

Gate Circuit

All of the decoding and fan-out trees in the memory are implemented using three-input current-mode gates like the one shown in Fig. 8. Two gates are included on each die as the photomicrograph of Fig. 9 indicates. Only three of the four input transistors shown are actually utilized, and a single layer of metal is used to interconnect the circuit. The gates are capable of driving a 50-ohm load with a typical delay of 1.35 nanoseconds. One die is bonded in each 14-lead flatpack and the total power per package is approximately 100 milliwatts.

PACKAGING

Since a signal experiences a delay of one nanosecond when traveling a distance of 6 inches down a transmission line on a printed circuit board made of epoxy glass, the physical layout of the system is

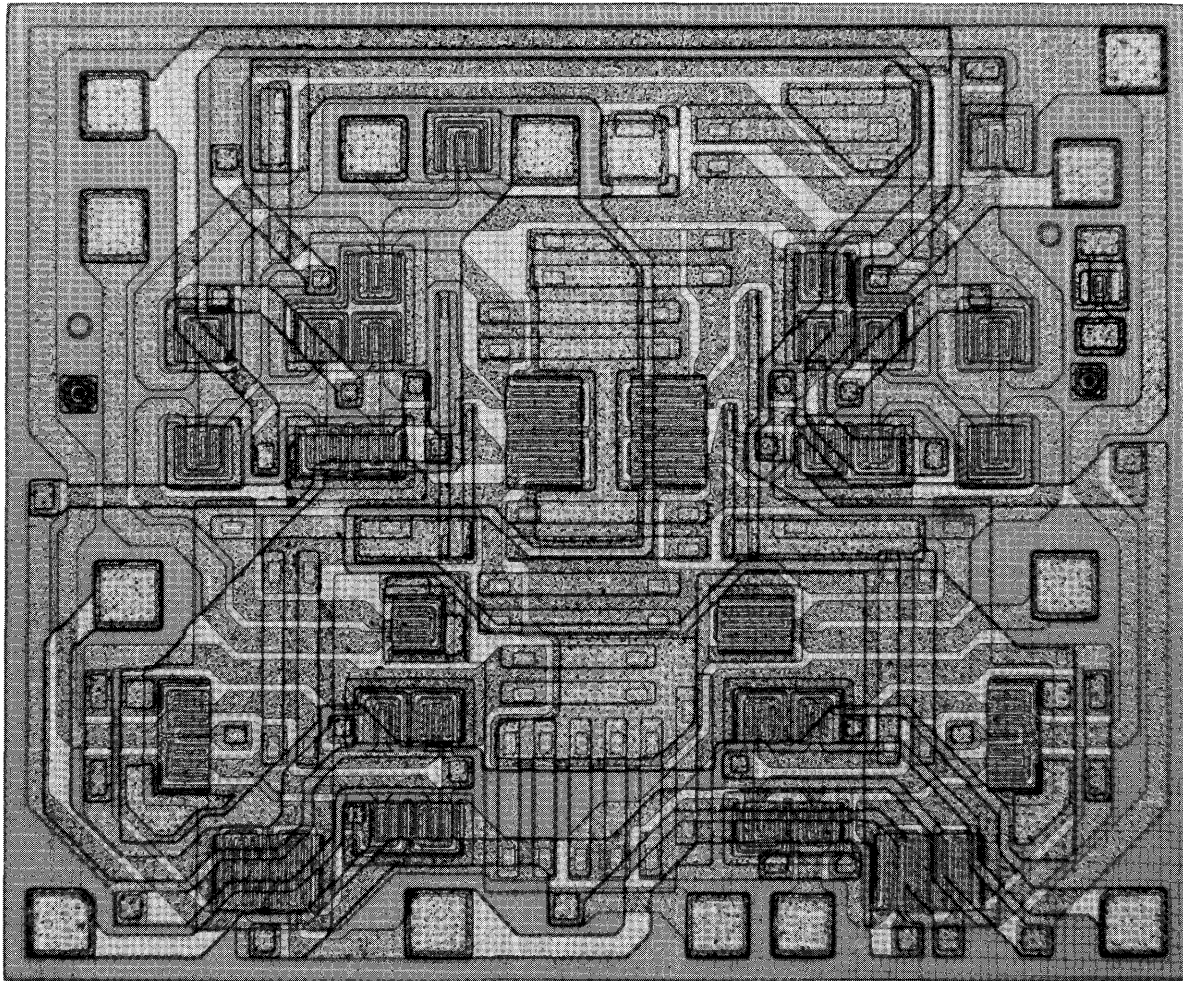


Figure 4. Photomicrograph of memory bit die.

carefully designed to minimize line lengths. In the final design, a distance of about 18 inches is traveled in each of the 17-nanosecond read cycles, resulting in a time loss due to signal propagation of some 3 nanoseconds compared to 14 nanoseconds of delay through circuits. This gives a ratio $\text{Line Delay}/\text{Circuit Delay} = 3/14 = 0.21$, and makes the ratio of line delay to total delay $3/17$ or 0.18. This is quite good since a reasonable ratio for line delay to circuit delay is up to $1/3$, or 0.33, making the ratio of line delay to total delay $1/4$ or 0.25. It should not be much higher for the following reasons:

1. Inasmuch as faster circuits are more expensive, it is inefficient to pay for circuit speed which is subsequently lost in interconnections.

2. The time taken for reflections to damp-out in a transmission line increases with line length. If lines are long compared to circuit delays, line characteristic impedances and line terminations have to be more accurate, resulting in extra cost.
3. The problem of mismatch due to branches in a transmission line is greater with longer lines.

Basic Packaging Philosophy

In a memory, the worst-case delay is paramount. Nothing is gained by making access time to one memory word less than access time to the worst-case memory word. A rule was formulated which

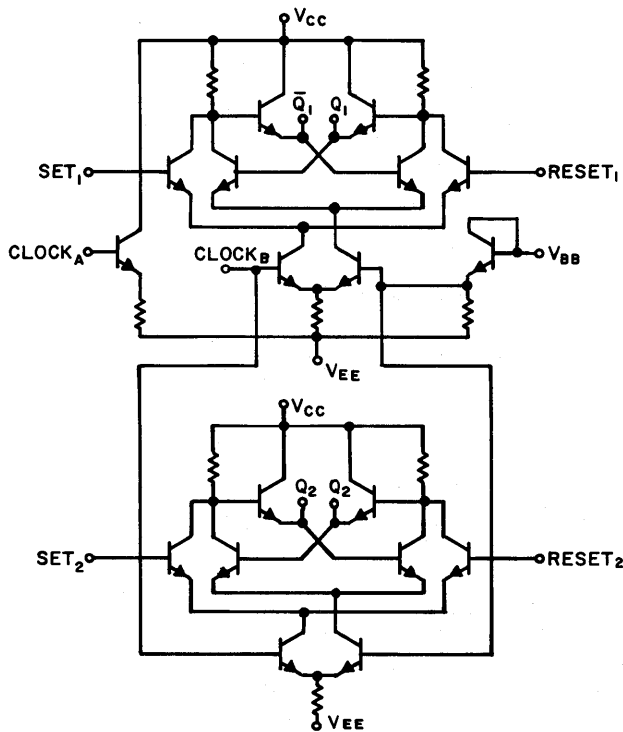


Figure 5. Schematic of register circuit.

was of great help in organizing the physical layout of the memory:

The fastest memory layout is that where access time to every word is the same, i.e., every path is a worst case delay path. This approach avoids timing problems by eliminating signal skew.

A 2-dimensional, 64-location storage array is represented in Fig. 10a. Assume that the word-select information enters the array at A and the output data leaves the array at B. If it is further assumed that signal propagation is restricted to the X and Y dimensions (no propagation along diagonals), it is clear that the distance from A to any storage location to B is a constant. For example, the length of path ACB is equal to the length of path ADEFB.

It is required that points A and B be on one connector at one edge of the memory. To meet this requirement, the memory array of Fig. 10a is folded at the centerline, GH, putting points A and B on opposite ends of a rectangle as shown in Fig. 10b.

As shown in Fig. 10c, there are always two paths between two points such as A and E. In address decoding, half of the address lines are sent each way and the final level of decoding performed in a two-

input AND gate located at the memory word. This is similar to the well-known coincident current decoding used in magnetic core stores. Write command and write data can travel either route from A to E equally well. Likewise, output data can take either route from E to B.

The foregoing refers only to the layout of the memory storage elements and the second level of decoding. Consideration is now given to the addition of address registers and first-level decoding required at the input to the memory and the output fan-in and registers required at the output of the memory. Figure 10d shows the location of the address registers, first-level decode, output fan-in, and output data registers. A typical set of paths through the memory is as follows:

1. Read initiate enters at the left edge of the memory.
2. Half of the first level decode information (X-decode) is sent up the left edge of the memory and horizontally across the array.
3. Half of the decode information (Y-decode) is sent vertically up through the array.
4. The output data is sent to the output fan-in gates which then drive the output data register.

System Implementation

To achieve equal path lengths from any word in the memory array to the output, it is necessary to send the output data from all storage locations to the far end of the card in the Y dimension, and from there back to the near end. By making the Y dimension very short, it is possible to simplify the wiring by not sending the output data to the far end of the array. Signal skew is still very small as long as the difference in path lengths is small compared to the total line delay plus circuit delay. This is achieved by placing the memory circuits on small cards in the third (Z) dimension. Making the cards removable from the mother card results in ease of maintainability.

The system packaging dimensions are somewhat fixed by a requirement that the memory be compatible with an existing larger system. Two of the overall system dimensions are limited to 4.5 and 6.875 inches, and cooling is specified as forced air.

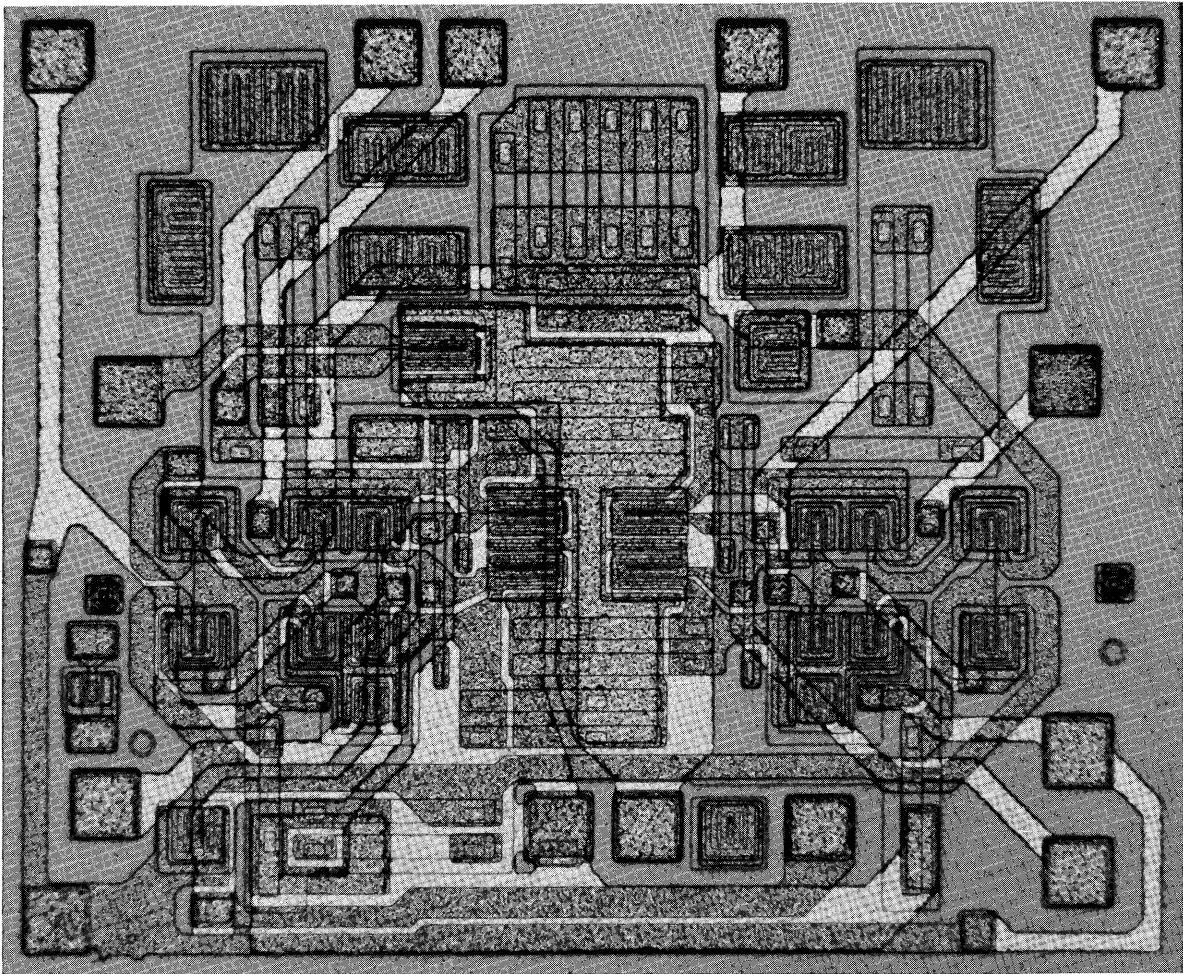


Figure 6. Photomicrograph of register circuit die.

A detailed discussion of cooling follows in another section.

A convenient division of the system into multiple groups of identical logic results in 12 multilayer printed logic cards interconnected by plugging into the 13 layer mother card as shown in Fig. 11. The mother card consists of seven signal and six voltage planes to form interconnections between the logic card connectors spaced on 0.2-inch centers.

The basic printed logic card, measuring 1.6×6.5 inches, interconnects three rows of up to 16 ceramic flat packages. This arrangement, shown in Fig. 12, results from the compatibility requirement and the ease with which the memory logic divides into sections. Three signal and three voltage planes are required. The inner signal plane is designed to be 55 ohms characteristic impedance, while the surface

conductors are designed to be 75 ohms. All critical path logic is routed, where possible, on the higher impedance surface conductors to reduce the circuit input capacitance loading effect. On the long lines, the distributed loading reduces the effective impedance to approximately 50 ohms. Conductor widths

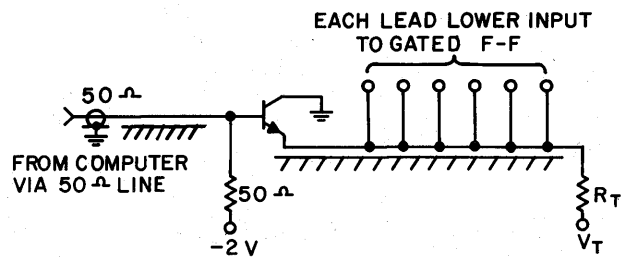


Figure 7. Method of distribution of clock pulses to register circuits.

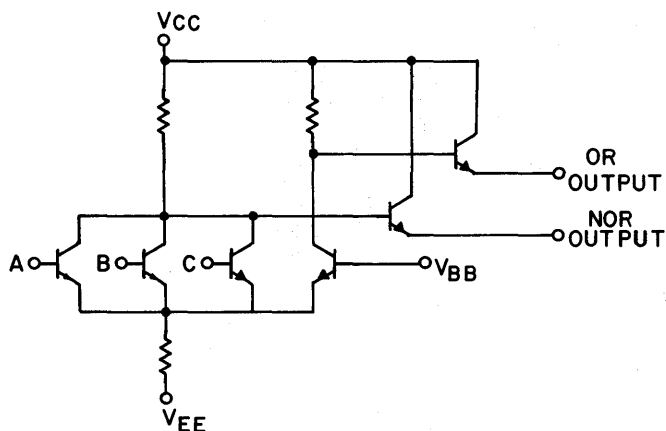


Figure 8. Schematic of three-input gate circuit.

are 6 mils on the inner plane and 8 mils on the outer planes. The tolerance on the conductor widths is $\pm\frac{1}{2}$ mil. The overall card thickness is 50 mils. Each circuit mounting pad has a plated-through hole to serve as a signal crossover when necessary and to add mechanical strength. The leads on the packages are bent flush with the bottom of the package and then reflow soldered on top of the multilayer card. This allows the packages to be mounted and replaced very easily.

Since memory is highly repetitive, division into sections can be made in many ways. The memory elements are divided into eight groups of eight words, each group being contained on one of the small logic cards. The 64 bits on each card are sub-divided into four groups as shown in Fig. 13.

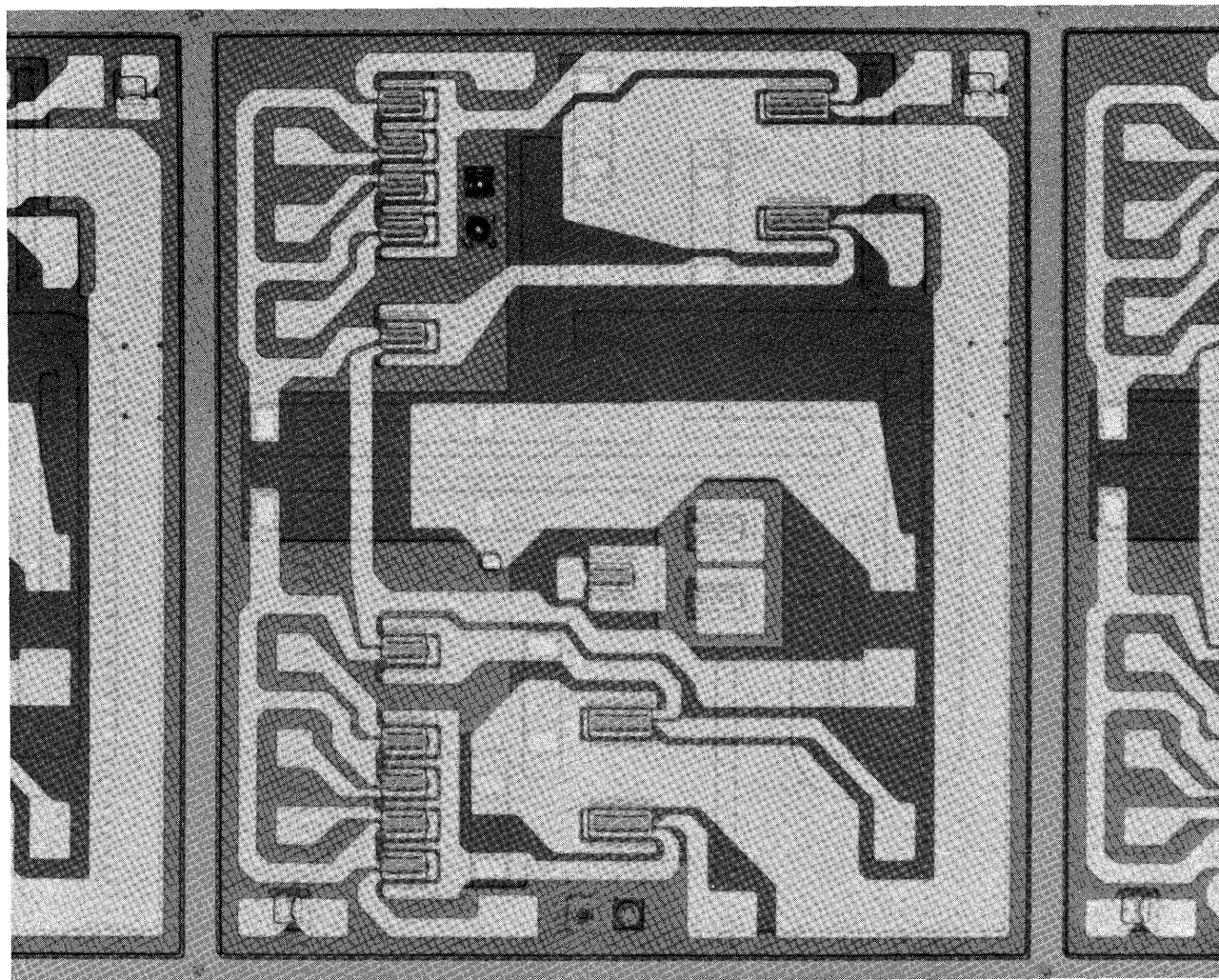


Figure 9. Photomicrograph of gate circuit die.

Each subgroup contains two words and the last level of read, write, and clear logic for both words.

Three other card types of like dimensions contain the input/output circuits. In keeping with the previously mentioned layout philosophy, the input logic enters the matrix of memory logic from one corner while all outputs are taken from the opposite corner.

leftmost side while the outputs are on the right side. Logic for these three card types divides naturally into read input/output, write address, and write data.

The read input/output card contains the read address register, the first-level read decode, the output data fan-in, and the output data register. Two identical read input/output cards are required because of the dual read channel requirement. Timing for enter-

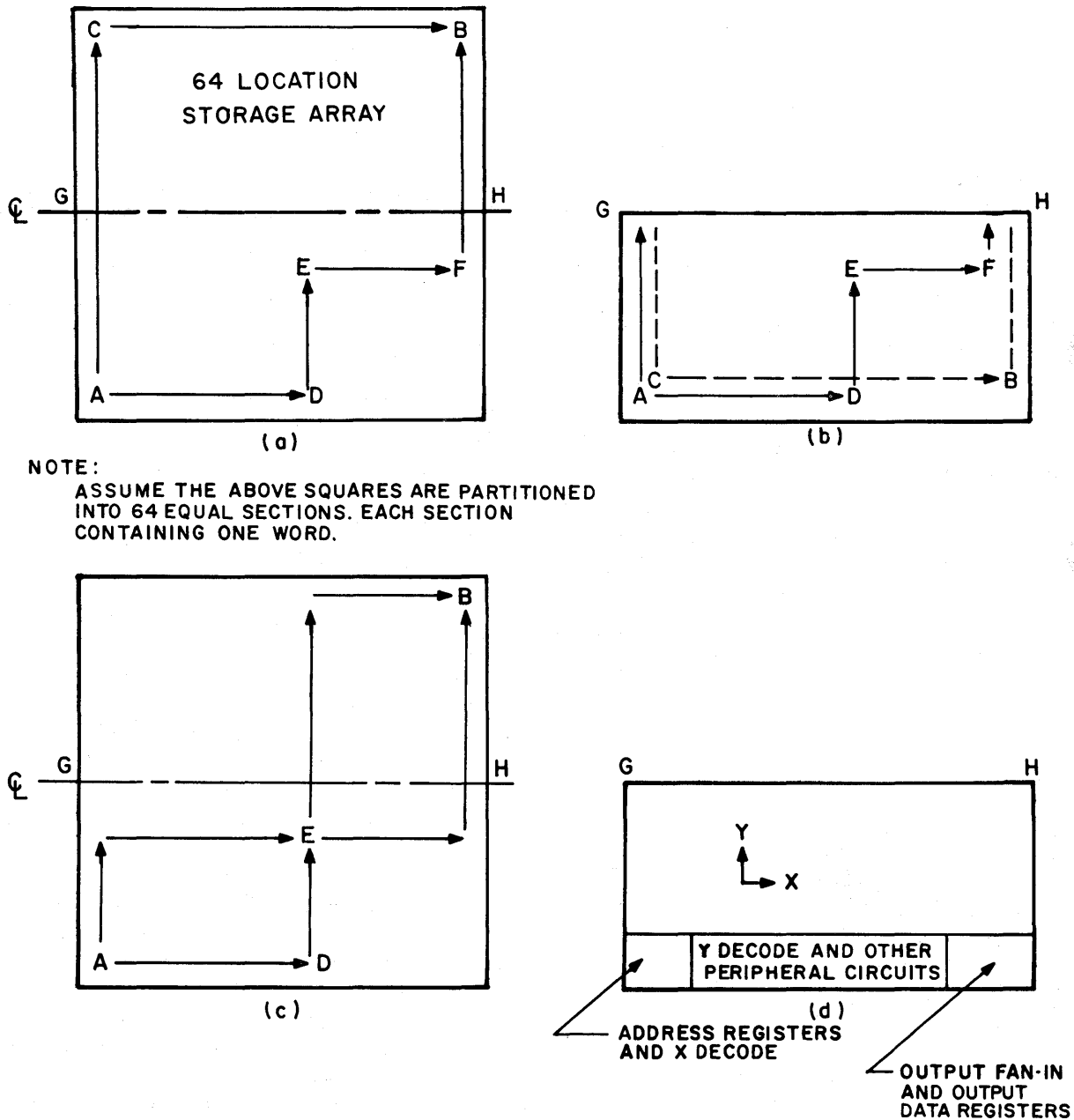


Figure 10. Two-dimensional storage array.

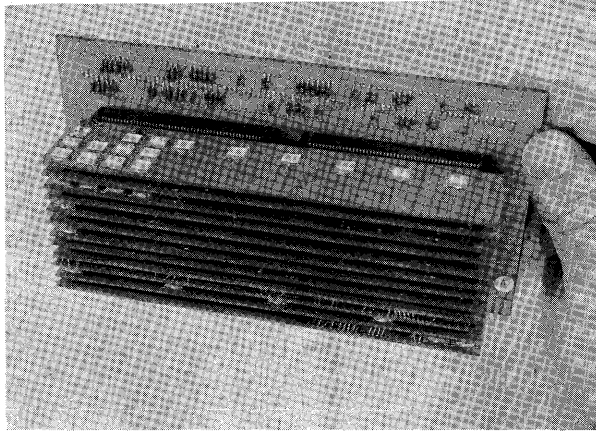


Figure 11. Scratchpad memory system.

ing output data into the output registers is obtained by suitably delaying the read initiate signal by the same time required to access the memory. This is accomplished through use of binary weighted lengths of printed circuit delay line suitably connected in series to form any desired delay to the nearest one-half nanosecond. This can be seen on the artwork for the inner plane of the read input/output card shown in Fig. 14.

The write address card contains the write address register, the first-level write decode, the first two levels of write command fan-out, and the first two levels of clear fan-out. As in the case of the read input-output card, the write address logic card has its inputs on the left side to equalize the length of all write paths. Write address information is entered under control of the read initiate signal. For proper

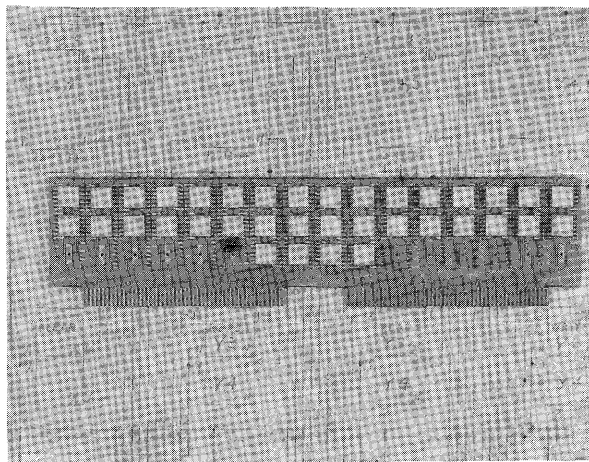


Figure 12. Printed circuit logic card.

timing, this signal is delayed through use of variable-length printed circuit delay lines.

The write data logic card contains the write data register and the write data fan-out gates. Each data bit enters a register located in the card's center and is then distributed to gates located at points across the card for an even distribution to the memory circuits.

In the preceding discussion, the many practical considerations leading to the exact dimensions of the overall memory package are described. It should be mentioned that a more analytical approach can be taken to determine if these dimensions are close to the values required to produce the minimum total path length through the memory. For minimum path length, an optimum ratio exists between the X and Y dimensions on the mother card (refer to Fig. 10d)

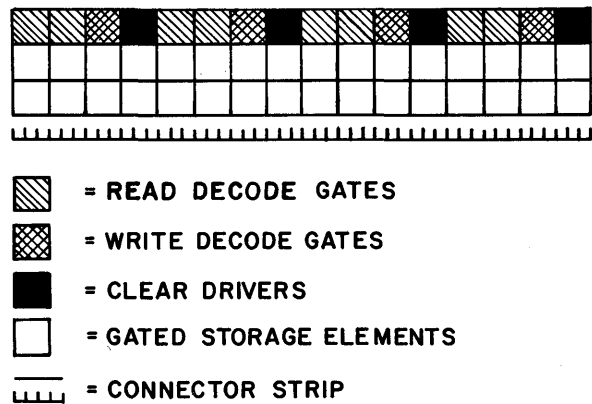
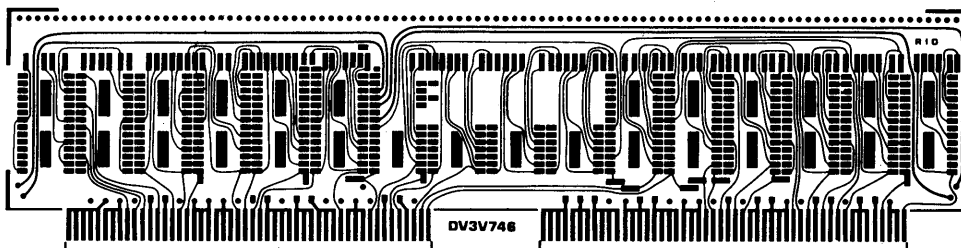


Figure 13. Subdivision of memory array logic.

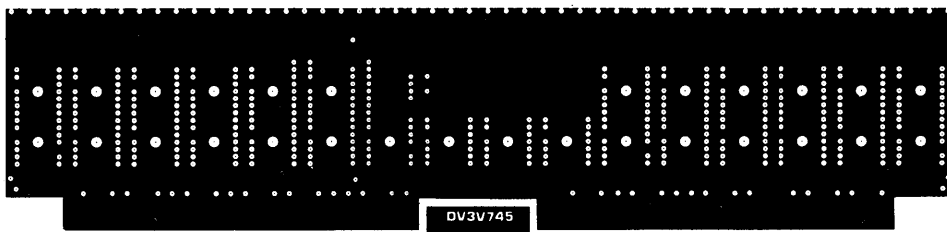
and the Z dimension on the plug-in logic cards. By relating both the total path length and the area required to contain the circuits to the X, Y, and Z dimensions, it can be shown that the optimum ratio of X:Y:Z should be 6:3:1. The ratio obtained using the actual package dimensions is 6:2.2:1.05. The fact that the optimum ratio is not used results in only 0.2 inches (40 picoseconds) of extra distance that must be traversed during each read access. This is insignificant compared to the total path length.

The 14-pin ceramic flat package is used as the circuit package because (1) sufficient leads are available to handle the required circuits, (2) the heat transfer characteristics are adequate as discussed in the Thermal Study section, and (3) the reflow solder attachment technique allows ease of assembly and repair.

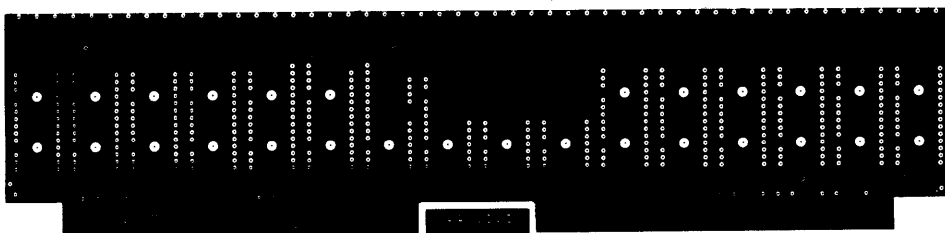
Separate terminating resistors are required since the circuits are designed for driving a 50-ohm trans-



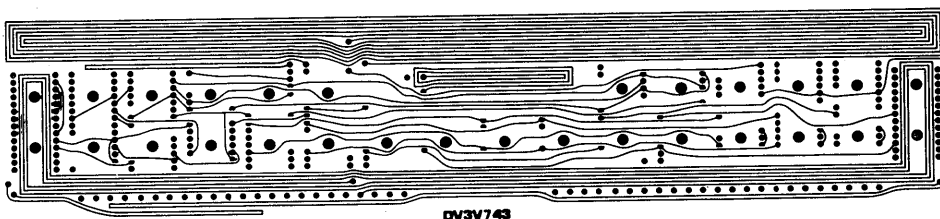
(a) SIGNAL PLANE, COMPONENT SIDE



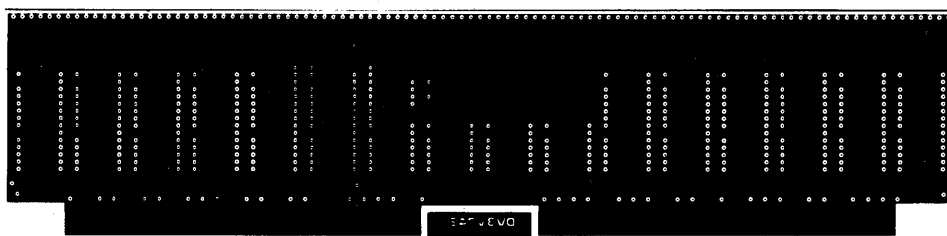
(b) VOLTAGE PLANE, V_{EE}



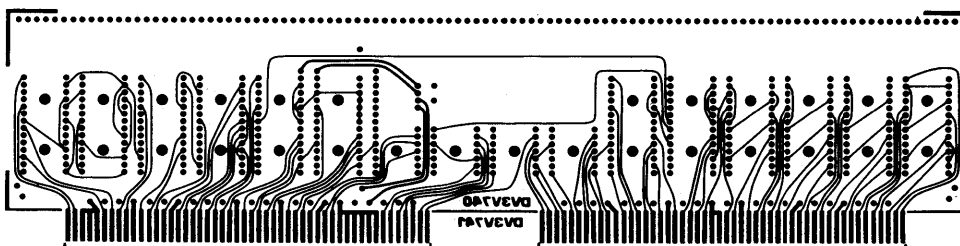
(c) VOLTAGE PLANE, V_{CC}



(d) SIGNAL PLANE, INTERNAL



(e) VOLTAGE PLANE, V_{BB}



(f) SIGNAL PLANE, PROBE SIDE

Figure 14. Read input/output logic card artwork.

mission line system. However, many lines are sufficiently short to be treated as lumped allowing a higher impedance termination. The 50-ohm terminations are resistive divider networks connected across the positive and negative supplies, giving a termination voltage slightly more negative than the negative logic level. Higher impedance terminations are single resistors returned to the negative supply. One-tenth watt carbon resistors are attached to the boards by reflow soldering in the same manner as the packages are attached. Lead lengths are kept to a minimum to reduce stray inductance and capacitance.

Provision is also made for attaching power supply bypass capacitors. In addition, the V_{CC} and V_{EE} voltage planes are separated by only two mils of dielectric resulting in a relatively large distributed capacitance throughout the memory.

The copper-clad epoxy material is selected for the thickness required to construct striplines of the desired characteristic impedance. Only normal control of etching is required to achieve the $\pm\frac{1}{2}$ mil tolerance on line width. Holes in the logic cards before plating are only 16 mils to reduce lumped capacitances and to allow routing of printed runs between holes on the inner plane.

Logic card connectors are required with a sufficiently large number of pins available to distribute voltages and signals across the entire width of the card to reduce transmission line discontinuities. A close card spacing is also required. These requirements are met using a double row strip connector having contacts on 50-mil centers. The connector thickness allows card spacing to be on 0.2-inch centers.

Prior to the design of the memory, a detailed study of stripline characteristics was made. This enabled the layout and fabrication of a system with predictable characteristic impedances and signal cross coupling. Signal cross coupling is limited to less than 5% in all cases. The high package density greatly reduces the number of crosstalk and termination problem areas.

Thermal Study

A thermal study was required to show feasibility of the proposed system packaging configuration. Due to the very fast circuit operation, steps were taken to reduce signal propagation times between circuits resulting in an increased power density. In addition, the cooling method was specified as forced air at

70°F and a pressure of 0.2 inches of water. This left thermal resistance, air flow pattern, and cooling surface area as variables.

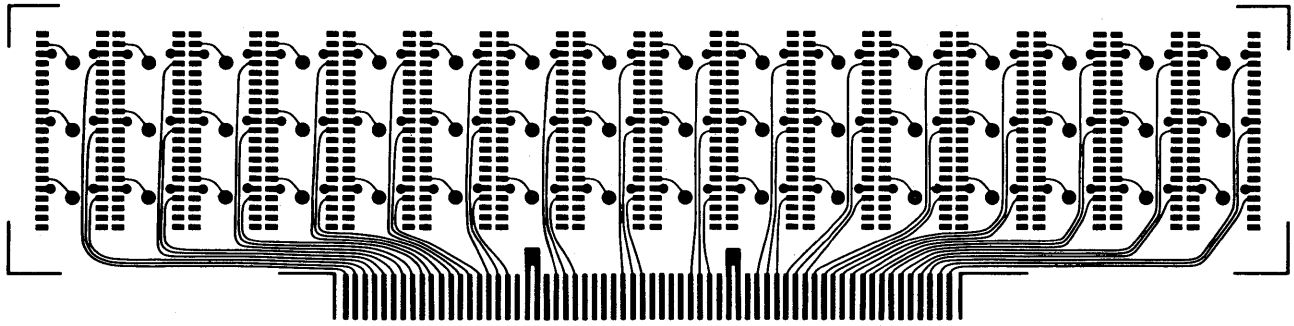
A total of 50°C junction to ambient drop was set as a design goal to limit the junction temperature to less than 75°C. Therefore, an anticipated maximum 0.3 watt per package limited the junction to ambient thermal resistance to less than 166°C per watt. A standard production type 0.25 by 0.25 inch ceramic flatpack was examined and found to have a junction to case thermal resistance of 30 to 40°C per watt depending upon the quality of the die bond. This is acceptably low if the case ambient thermal resistance is 125°C per watt or lower.

To properly simulate the actual system's environment, multilayer thermal test cards (Fig. 15) were constructed which contained voltage planes in the same manner as the proposed system. Circuits mounted on these cards contained an isolated diode whose forward voltage drop could be measured as a function of temperature. This is shown schematically in Fig. 16. During testing, three fully populated cards were spaced on 0.2-inch centers and subjected to 25°C air under a blower pressure of 0.2 inches of water. Temperature measurements were made on the center card.

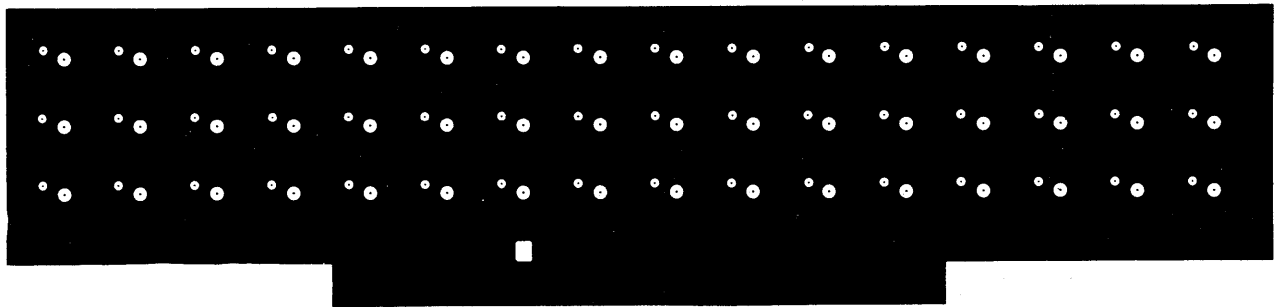
Four package attachment techniques were explored. In each case, the package leads were attached by reflow soldering to the test card. In the first test, each package was mounted with the bottom or circuit side of the package closest to the card. The test measurements showed about 190°C per watt thermal resistance with an insignificant variation along the length of the card. The second test involved a simple inversion of the package placing the chip side of the package away from the card and in the air stream. The result was an improved 167°C per watt. Though this mounting technique was acceptable, others were explored.

To effect further improvement, the packages were heat-sunked to the card to gain cooling area improvement. A plated-through hole was placed beneath each package which was connected to a copper voltage plane on the opposite side of the card. Heat transferred readily to all parts of the card thereby using both sides of the card as cooling surfaces. A nonsetting thermal joint compound was applied beneath each package during assembly. The result was a thermal resistance of 131°C per watt.

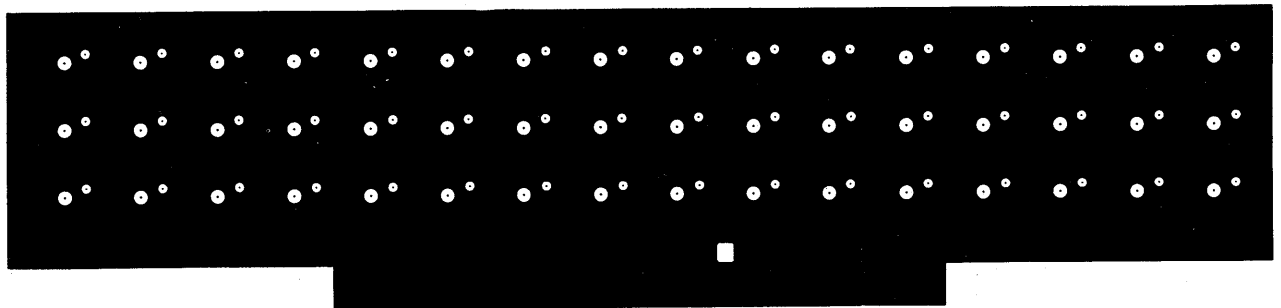
The second heat-sinking technique involved spe-



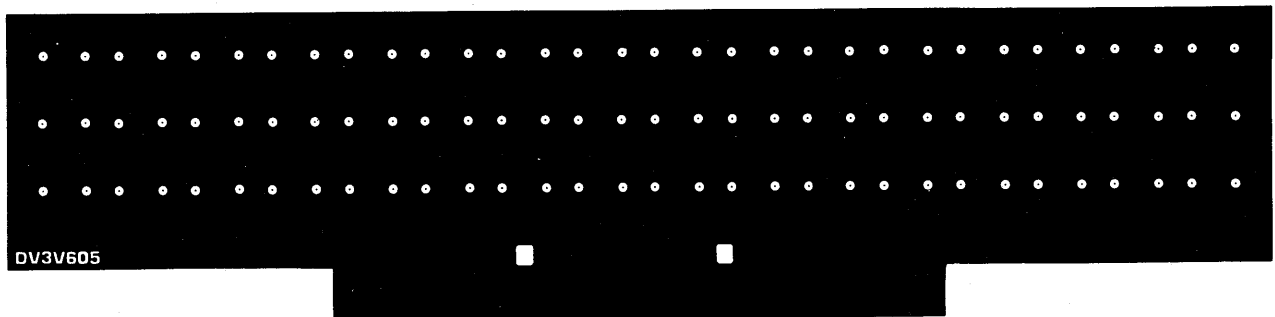
(a)



(b)



(c)



(d)

Figure 15. Thermal test card.

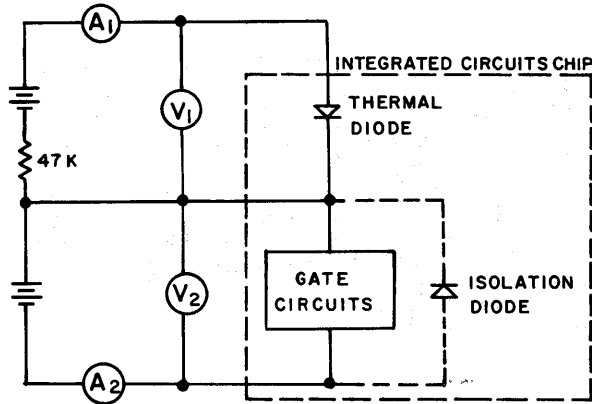


Figure 16. Thermal test circuit.

cially prepared circuit packages with metallized bottoms which were soldered to the plated-through holes with a low temperature solder. Though requiring a more involved attachment technique, test results showed a significant improvement at 88°C per watt.

The thermal joint compound is used in the final

system since it provides a margin of safety and presents no unusual assembly problems.

TEST RESULTS

Partially Populated Model

The printed circuit boards, as described in the Packaging section, are designed and fabricated for a 64-word memory of 8 bits per word. It is felt that four paths through the memory are sufficient to test the memory circuits and interconnections. The memory boards are populated in a manner such that four words of two bits each can be accessed for both read and write. Each circuit node in the four paths is fully loaded with fan-out so that worst-case circuit delays are present. The block diagram of Fig. 17 illustrates the circuits included in the partially populated model. The circuits are positioned such that the worst-case path lengths are encountered. Small variations in path length exist even though the memory is designed to minimize this effect. The difference

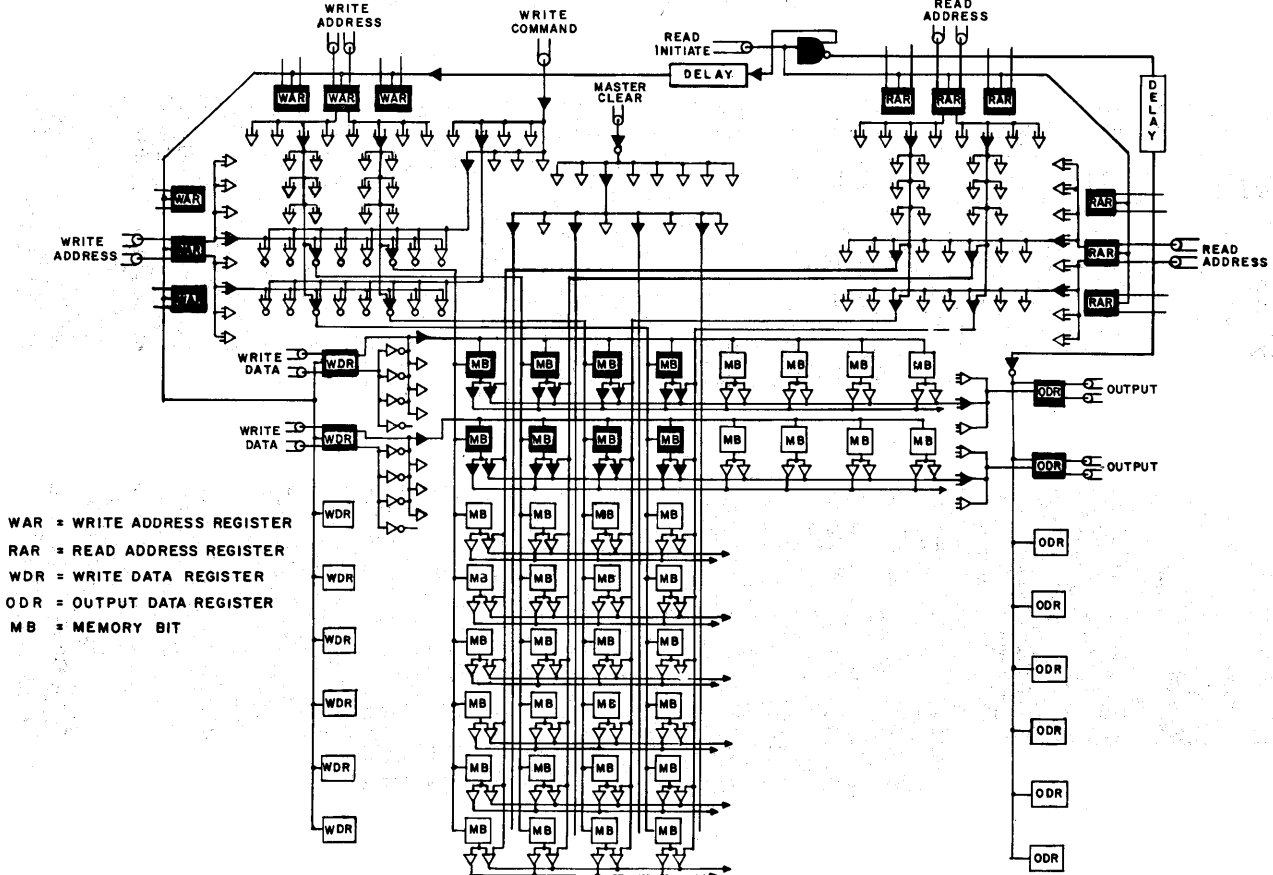


Figure 17. Partially populated model.

between the longest and shortest paths is in the order of 0.5 nanoseconds.

Worst case conditions on the following parameters are important in the operation of the memory:

- Circuit loading
- Stripline delay
- Crosstalk
- Reflections
- Power supply tolerance and noise
- Junction temperature

As discussed above, the worst-case circuit loading and stripline delays are included in the partially populated model.

The use of partially populated, full-size system boards also allows a fairly thorough evaluation of crosstalk and reflection problems. As mentioned earlier, the striplines are designed with the goal of keeping crosstalk below a certain level. The test results indicate that this effort is very successful since even the longest lines have crosstalk of well under the design goal of 5%. The memory is overdesigned in this respect since a 1-nanosecond risetime is assumed in the design, while a 2-nanosecond risetime is actually achieved in the system. All lines of under 2 inches in length are treated as lumped and a 180-ohm output resistor used for the gate load. All lines over 2 inches are treated as distributed and terminated in 50 ohms. As previously described, the actual striplines are designed for a 75-ohm characteristic impedance in areas of distributed load so that the loaded line is close to the desired 50 ohms. The reflections due to connectors, loads, and terminations are also well under 5% of the total signal amplitude.

The positive power supply can be varied by $\pm 10\%$ and the negative supply by $\pm 20\%$ before an error is detected. Also, the power supplies are almost entirely free from noise resulting from current transients. This is due to the use of the thin 2-mil dielectric between voltage planes and miniature tantalum capacitors distributed around the cards. It is felt that the results achieved in the partially populated model are more than satisfactory and that no problems will arise when the remainder of the memory is populated.

Since the full power dissipation is not present in the memory, worst-case temperature effects cannot be determined. The individual circuits have been tested at temperatures in excess of those predicted by the thermal studies previously described, how-

ever, with no adverse effects on the switching characteristics. The DC circuit characteristics are also within specification at the elevated temperatures. On the basis of these tests, no problems are anticipated in regard to higher junction temperatures in the full system.

Memory Exerciser

The memory exerciser is designed to test the memory with various patterns of read addresses, write addresses, and write data. Counters containing two stages are used to determine the read and write addresses. The two counters count in opposite directions and can be preset to any value. A third stage on the write address counter switches between two patterns of write data. The two patterns are simply mechanical switches which can be set to ONE or ZERO independently. The pattern bit in the counter changes state only after all four memory locations have been selected. This results in the pattern A information and the pattern B information being written into all four locations alternately. The exerciser compares the actual information obtained during a read cycle to the expected data and stops the machine if an error occurs.

Memory Operation

Figure 18 shows the read initiate pulse supplied by the memory exerciser, the memory bit output at the point of the tied emitter OR, and the output of the output data register. Thirteen nanoseconds are required from the time the read initiate clocks the read address into the register until the stored information is available at the input to the output data

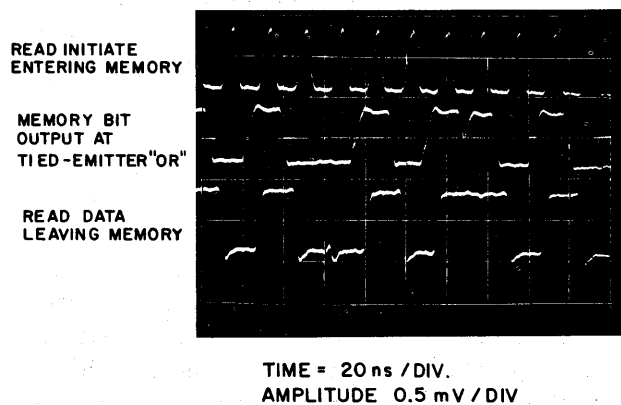


Figure 18. Memory waveforms.

register. The delayed read initiate arrives at the output data register one nanosecond later and clocks the stored information into the register. The data is available at the terminals of the memory 17 nanoseconds after the start of the read initiate pulse.

It is important to note that the memory bit output is directly compatible with the current mode logic and no amplification is required. This eliminates the usual memory noise problems and attendant recovery time periods allowing the extremely fast memory speeds. This is simply an extension of the same technology used in the remainder of the computer applied to the implementation of a high speed bank of independently addressed registers to be used in various critical points in the computer. A later section gives examples of the use of a memory of this type.

In the read-decision-write cycle, a delayed read initiate pulse clocks the write address and write data into the appropriate registers. The exerciser then supplies a write command pulse 15 nanoseconds after the read is completed. The write operation is completed approximately 8 nanoseconds after the start of the write command pulse.

In a more general application, the write command would be used to clock the write address and write data into the registers. This is a longer delay path and 10 nanoseconds would be required to perform a write operation.

The memory exerciser contains no bypass route in case the read address and write address are identical during a cycle. If this occurs, the old information is obtained during the read operation since the read is completed before the memory bit changes state (refer to Fig. 2). The memory bit changes state during the first portion of the next read cycle just prior to the time the address information filters down to the memory bit read gate. The error detection circuitry is designed to account for this type of operation.

The only limitation placed upon simultaneous read and write operations is that the memory bit cannot be accessed for a read operation during the period of time (approximately 2 nanoseconds) when the memory bit is actually changing state. As long as the system designer knows whether the memory bit is accessed before or after the change of state, he can use the memory output accordingly. This allows a large degree of flexibility on the part of the system designer.

It should be noted that the registers at the inputs and outputs to the memory are included for generality only. In most cases, these registers are contained in the logic portion of the computer and can be eliminated from the memory, assuming the proper timing can be guaranteed. This, of course, allows faster operation of the memory.

In certain instances, cycle overlap could be used to decrease the read and/or write periods. The actual read access time and/or write time would remain the same; however, the rate of operation would be increased by taking advantage of the read address, write address, and input data storage time in the various decode and fan-out trees.

SYSTEM APPLICATIONS

The memory described thus far is designed for a particular application requiring a read-decision-write cycle and 64 words of 8 bits per word. As previously mentioned, other modes of operation for different applications are possible. The memory capacity can also be altered quite easily.

The number of bits of storage can be increased in several ways. A modular approach can be used by connecting 64×8 memories in parallel or the memory boards can be redesigned to accept the larger number of bits. The modular approach is particularly applicable to the distribution of small memories of various sizes throughout a large computer. It is possible to construct a 64×32 memory using either of the above approaches with a cycle time of approximately 20 nanoseconds. The extra 3 nanoseconds result from one stage of logic plus line delay.

The memory has all of the usual applications of scratchpad memories, but its two read channels suggest other special applications.

The arithmetic unit of a computer operates on two words at a time. A dual read operation could provide both words to the arithmetic unit in only one memory access time. The simultaneous write feature would allow the memory to be written into at the same time the read operations were taking place. The data being written could come either from the main memory or the arithmetic unit.

In some computers, a separate adder is used for index and dynamic memory relocation operations. A memory with dual read capability could provide

the index quantity and relocation coefficient to the adder simultaneously. The adder would then add the above two quantities to the address in one step.

The dual read operation could also be used to provide both halves of a word for double word length operations.

All of the above applications are used to reduce the effective memory access time by one-half.

CONCLUSIONS

This paper describes an extremely high-speed memory that is very flexible in application. The physical size required to implement a flip-flop memory of this speed can only be obtained using densely packaged integrated circuits. The use of a storage element operating at standard logic levels contributes greatly to the high speed and allowed the development of this memory in the relatively short period of 10 months. The memory speed, flexibility, and development time are all improvements over prior attempts to implement under-100-nanosecond memories using magnetic techniques.

ACKNOWLEDGMENTS

The authors extend their thanks to all of the persons in the Integrated Circuit Research and Development group at Motorola who contributed to the design and construction of the memory described in this paper. The efforts of Howard Hannah, Robert Hawn, Dave Turcotte, and Anthony Ziolkowski are particularly appreciated.

REFERENCES

1. R. P. Shively, "SMID: A New Memory Element," *Fall Joint Computer Conference*, vol. 27, part 1, Spartan Books, Washington, D. C., 1965, pp. 637-47.
2. H. A. Perkins and J. D. Schmidt, "Integrated Semiconductor Memory System," *ibid*, pp. 1053-64.
3. J. R. Burns et al, "Integrated Memory Using Complementary Field-Effect Transistors," 1966 International Solid-State Circuits Conference, *Digest of Technical Papers*, vol. IX, pp. 118-19.
4. G. B. Potter and J. Mendelson, "Integrated Scratchpads Sire New Generation of Computers," *Electronics*, vol. 39, no. 7, pp. 118-26 (Apr. 4, 1966).

SONIC FILM MEMORY *

H. Weinstein, L. Onyshkevych, K. Karstad and R. Shahbender

*Radio Corporation of America
Princeton, New Jersey*

INTRODUCTION

The sonic film memory represents a novel approach to the storage of digital information. Thin magnetic films and scanning strain waves are combined to realize a memory in which information is stored serially. The remanent property of magnetic films is used for nonvolatile storage. The effect of strain waves on magnetic films is used to obtain serial accessing. This effect is also used to derive a nondestructive read signal for interrogation.

The concept of a block-oriented random access memory¹ (BORAM) was recently introduced. In such a memory, information is stored in blocks with random access to blocks being accomplished at electronic speed. Information in a block is retrieved serially at bit rates compatible with computer speeds. The sonic film memory represents one possible realization of a BORAM.

SYSTEM DESCRIPTION

The simplest configuration for a sonic film memory block^{2,3} consists of a glass tube with a magnetic film coating. A conductor threads the tube, an ultrasonic transducer and an ultrasonic absorbing termination are attached to the ends, as shown in Fig. 1.

* The work described in this paper is partially supported by the USAERL under contract numbers DA36-039-AMC-03248(E) and DA28-043-AMC-01392(E).

The magnetic coating has square loop properties and is originally saturated circumferentially along the entire length in one direction. To enter information serially into the block, a strain wave is launched along the tube and propagates from the transducer end to the termination. At the termination, the wave is completely absorbed. As the wave propagates along the tube, it reduces the switching threshold of the strained region. A pulse of current applied to the conductor linking the magnetic coating will cause flux reversal only in the strained region permitting the selective writing of information.

Nondestructive serial retrieval of the stored information is effected by propagating a strain wave along the rod. This produces a reversible change in the

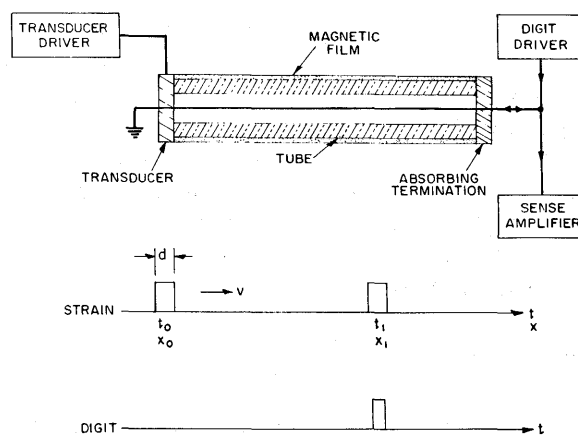


Figure 1. Sonic film memory realization.

magnetization of a strained region. This change induces a voltage in the linking conductor whose polarity depends on the remanent state of the flux corresponding to the stored information.

Figure 1 shows that the region in space occupied by the strain wave determines the dimensions of a "bit." The spatial dimension d , shown in Fig. 1, is given by: $d = vt$, where v is the velocity of sound, and t is the time duration of the strain wave.

To enter information, it is necessary that the strained magnetic region complete its switching in a time which is a fraction of the time duration of the strained pulse. For high bit densities, say in excess of 50 bits to the inch, this implies a switching time of the order of 10 nanoseconds.

Assuming that the strain wave causes a reduction in the magnetic threshold of only a factor of 2, it is clear that switching must be accomplished with relatively modest overdrives. The requirement of high-speed switching, with relatively modest overdrives, rules out the possibility of using isotropic magnetic materials for the storage medium. However, thin magnetic films with uniaxial anisotropy have the required property of very fast switching, of the order of 2 nanoseconds, with drives of the order of the anisotropy field.⁴

For the memory configuration shown in Fig. 1, and high bit densities, it is necessary that the magnetic coating have uniaxial anisotropy and be capable of fast switching. Such coatings with a circumferential easy axis and an axial hard axis are in use for random access memories.⁵

STRAIN DEPENDENCE OF MAGNETIC CHARACTERISTICS OF FILMS WITH UNIAXIAL ANISOTROPY

The gross magnetic properties of a planar film with uniaxial anisotropy may be described by specifying the direction of the anisotropy (easy axis), the magnitude H_k of the anisotropy field, and H_c the easy axis coercive field. The dependence of the anisotropy direction and anisotropy field on uniaxial strains applied at an arbitrary angle with respect to the easy axis may be derived analytically.⁶⁻⁸ The results of these derivations are:

$$\tan 2\theta_r = K \frac{\sin 2\phi}{1 + K \cos 2\phi}$$

$$h_k = H_k/H_k^0 = (1 + K^2 + 2K \cos 2\phi)^{1/2}$$

$$h_k = \sin 2\phi / \sin 2(\phi - \theta_r)$$

where ϕ = angle between the strain axis and the unstrained easy axis,

θ_r = the angle through which the strain rotates the easy axis (referred to the unstrained easy axis),

H_k^0 = the strain-free anisotropy field,

H_k = the effective anisotropy field in the strained film,

K = strain parameter = $\frac{S}{H_k^0} \frac{\Delta H_k}{\Delta S}$ for $\phi = 0$ or $\pi/2$, and

S = strain.

Figures 2 and 3 are plots of the above equations.⁸ In Fig. 2, θ_r is plotted as a function of K with ϕ as a parameter. In Fig. 3, H_k is plotted versus K with ϕ as a parameter.

Inspection of Fig. 2 shows that if the strain is applied along the easy axis ($\phi = 0^\circ$), the anisotropy direction is unchanged for all values of strain (tensile or compressive). On the other hand, if the strain is applied at right angles to the easy axis ($\phi = \pi/2$), the anisotropy direction changes discontinuously at $K = 1$ from $\theta_r = 0^\circ$ to $\theta_r = \pi/2$.

Inspection of Fig. 3 shows that H_k can be increased or decreased by application of strain. For a positive magnetostriction coefficient and a tensile strain, K is

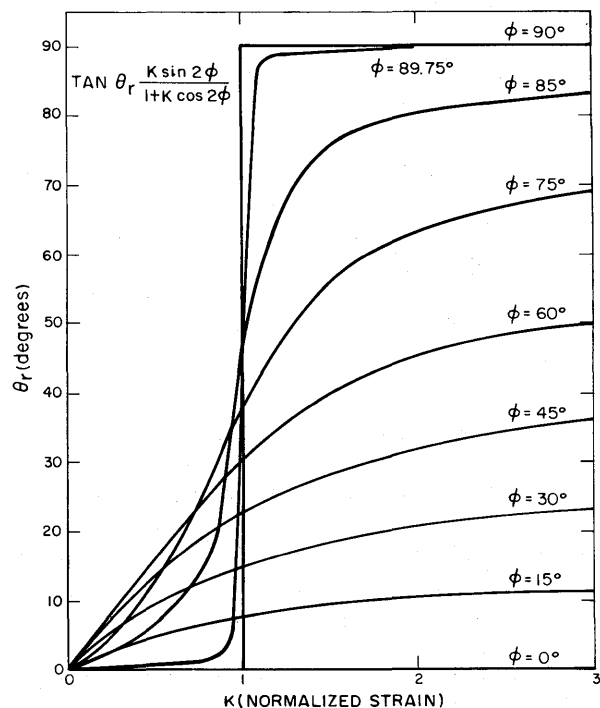


Figure 2. Axis rotation angle θ_r vs strain parameter K .

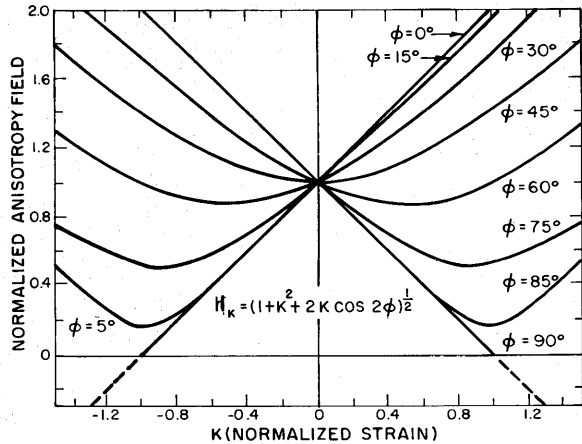


Figure 3. Normalized effective anisotropy field vs strain parameter K .

positive (a compressive strain corresponds to K negative). For this case the normalized anisotropy field H_k is reduced by the application of a compressive strain along the easy axis ($\phi = 0$) or a tensile strain along the hard axis ($\phi = \pi/2$). From Fig. 3 it is seen that for values of $K > 1$ and $\phi = \pi/2$ the $H_k < 0$. This is a result of the discontinuous rotation of the easy axis as indicated in Fig. 2.

Experimental data on planar thin films,⁸ and films with cylindrical symmetry⁹ are essentially in agreement with these theoretical predictions.

For highly magnetostrictive iron-nickel films the experimental data indicates that a strain of 10^{-4} is required to produce discontinuous rotation of the easy axis. Similarly, a reduction in the anisotropy field by a factor of 2 is realized with a strain of 10^{-4} . To accomplish sequential selection in a memory block, either the rotation of the easy axis with strain, or the reduction of the anisotropy field with strain, or a combination of both effects may be used. In either case, the required strain is of the order of 10^{-4} , and it is necessary to generate this strain at the film. Use of magnetic compositions with higher magnetostriction permits a reduction in the required strain amplitude.

STRAIN SWITCHING OF THIN MAGNETIC FILMS

The switching thresholds for thin magnetic film are given by the well-known astroid shown in Fig. 4. For an unstrained element the thresholds are determined by the value H_{ku} and the outside astroid

in Fig. 4a. For a strained element, the threshold is given by H_{ks} shown by the inside astroid in Fig. 4a. This assumes that the strain is applied along the easy axis or along the hard axis and is of a magnitude so as not to produce discontinuous rotation. Consider the points P1 and P2 in Fig. 4a. It is clear that a hard axis bias in combination with either a positive or a negative easy axis drive will cause switching of a strained element but will not cause switching of an unstrained element. Thus a coincidence of an easy axis drive with a strain pulse will cause switching at the local strained region.

Alternatively, the rotation of the easy axis due to the application of strain may be utilized to enter information into the memory. In this case, the strain rotates the easy axis and an easy axis drive completes switching of the strained region. This is illustrated by the astroids shown in Fig. 4b. Bipolar easy axis drives are required to produce both remanent states. In an actual film, a combination of both rotation and reduction in the anisotropy field is probably the cause of switching.

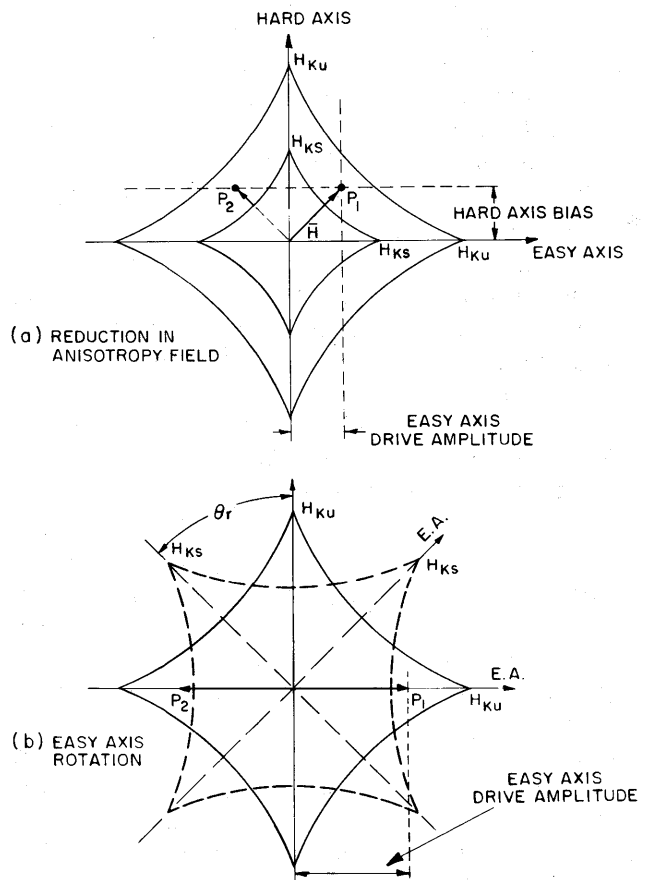


Figure 4. The switching astroid.

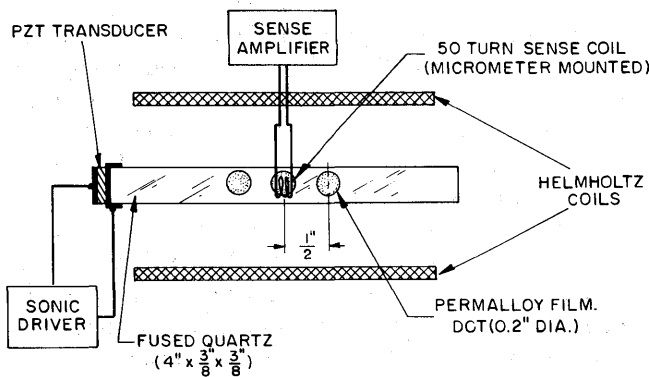


Figure 5. Memory device geometry.

MEMORY CELL TEST DATA

Static Magnetic Fields

The storage system selected for initial testing consists of thin permalloy films deposited on a fused quartz substrate. The substrate measures $4 \times \frac{3}{8} \times \frac{3}{8}$ inches and has a PZT transducer bonded to one end surface as shown in Fig. 5. The PZT transducer operating in the longitudinal mode is used to generate strain pulses in the substrate. This is accomplished by exciting the transducer at its resonant frequency of 4 Mcps, from an external generator with a modulated sine wave voltage.

The thin film storage elements have a composition of 60:40 nickel-iron and a diameter of 200 mils. The spacing between the film spots is 0.5 inches to eliminate magnetic coupling between the spots from interfering with the measurements. The hysteresis loops measured under static strains for samples of this composition indicated a uniaxial anisotropy together with a relatively high strain sensitivity. The substrate with the magnetic films is inserted in the mounting jig shown in Fig. 6.

For the device shown in Fig. 5, the strain pulse propagates in a direction along the hard axis of the magnetic films. Helmholtz coils provide bipolar magnetic fields of approximately 5 Oe in the easy direction during writing and are used to establish the original direction of magnetization of the films. A 50-turn coil is positioned close to a film element and is used to detect the output sense signal. A strain pulse packet is generated by the transducer and sequentially scans the magnetic films. Figure 7 shows the voltage applied to the transducer. The first signal is the voltage applied by the generator to the transducer and the second waveform is due to the

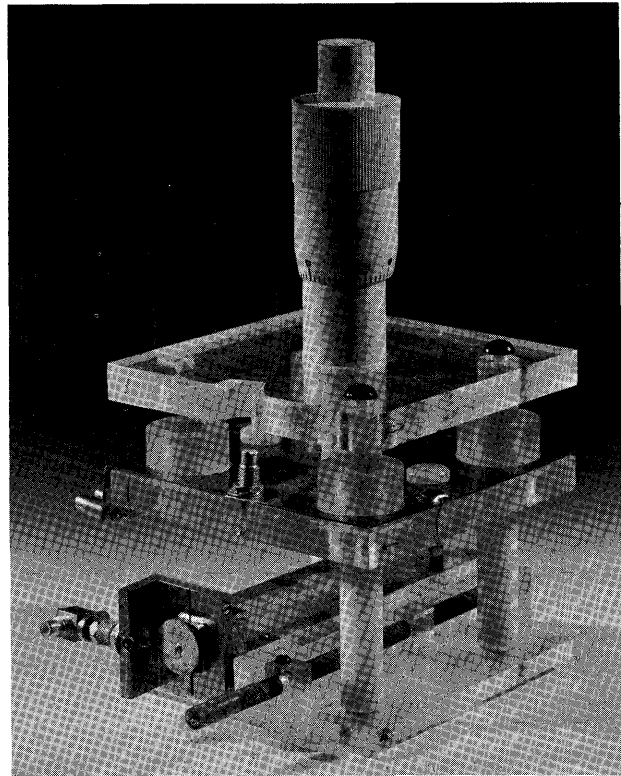


Figure 6. Sonic film memory rod mounted on drive jig (side view).

strain wave being reflected from the far end of the quartz substrate and impinging on the transducer. The lower trace in Fig. 7 shows the waveform obtained from the sense coil. This waveform is expanded in Fig. 8 for both binary remanent states in the films. As can be seen in Fig. 8, the sense signals are bipolar in nature. They are truly nondestructive for limited strain amplitudes.

The magnitude of the sense signal was found to be linearly dependent on both the strain amplitude and the diameter of the film spot. When a static easy axis magnetic field is maintained during the sense operation, the magnitude of the sense signal diminishes. For a field of 7 Oe, the sense signal is reduced to zero.

To switch a film with a given strain amplitude, a threshold value for the static magnetic field exists. The higher the amplitude of the strain pulse, the lower is the threshold value for causing magnetic reversal. For transducer excitation of 100 volts peak value, the magnetic threshold is 2.8 Oe. For a sonic drive amplitude of 50 volts peak value, no change in magnetization is observed when the magnetic field is reversed. This demonstrates that a coin-

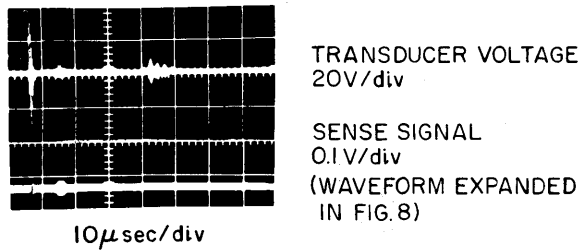


Figure 7. Sonic drive and sense signal.

cidence of a magnetic field and strain pulse is required to cause switching in the magnetic film.

To further demonstrate the need for coincidence between strain pulses and magnetic field pulses to produce switching, dynamic "Write-Read Disturb" testing was undertaken.

Dynamic Testing

For this test the sense coil of Fig. 5 is used to apply the magnetic bit field. A common clock provides synchronization between the sonic and magnetic pulses. The sonic drive program consists of a doublet of sinusoidal bursts as shown in the timing diagram of Fig. 9. The first burst is used for writing information into a memory element. The second burst is used to read out the information. The magnetic field pulse is applied via the sense coil in coincidence with the strain burst arriving at the storage element. The upper trace in Fig. 10a shows the voltage waveform applied to the transducer for a write-read sequence. The lower trace illustrates bipolar magnetic digit drives (one and zero) timed to coincide with the arrival of the write strain pulse at the test storage location. Figure 10b illustrates the disturb condition. Only magnetic energy is applied to the memory cell under test without a write strain pulse. Figures 10c and 10d show the

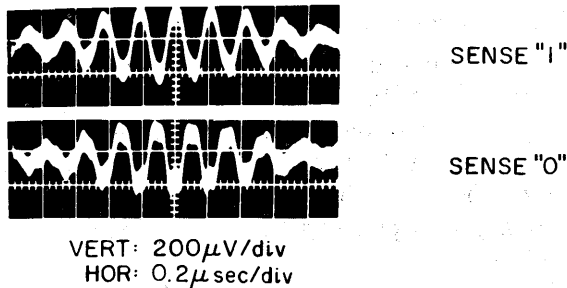


Figure 8. Expanded sense signals.

sense signals corresponding to the write and disturb conditions, respectively. For Fig. 10c the sense signals are bipolar corresponding to the binary state. Figure 10d shows the sense signals obtained by disturbing the binary one state by both positive and negative digit pulses. As can be seen, the digit pulses produce very little disturb effects.

For this set of dynamic tests, distinct thresholds were observed for both the strain and magnetic drive amplitudes. However, the margins of operation for both the strain and magnetic field pulses were narrower than in the case of static magnetic testing.

Unipolar Strain Pulse Testing

For this set of tests, films with an iron-nickel-cobalt composition having very high magnetostriction are used. The elements, with a rectangular shape, measure 0.1×0.15 inches and are approximately 1000 Å thick. These films are deposited on conventional microscope glass slides measuring $1 \times 3 \times 0.04$ inches. A shear mode PZT transducer, resonating a 1.5 MHz, is bonded to one end of the glass slide, a fine layer of viscous material is used for damping ultrasonic reflections. Figure 11 shows the geometrical arrangement of the magnetic

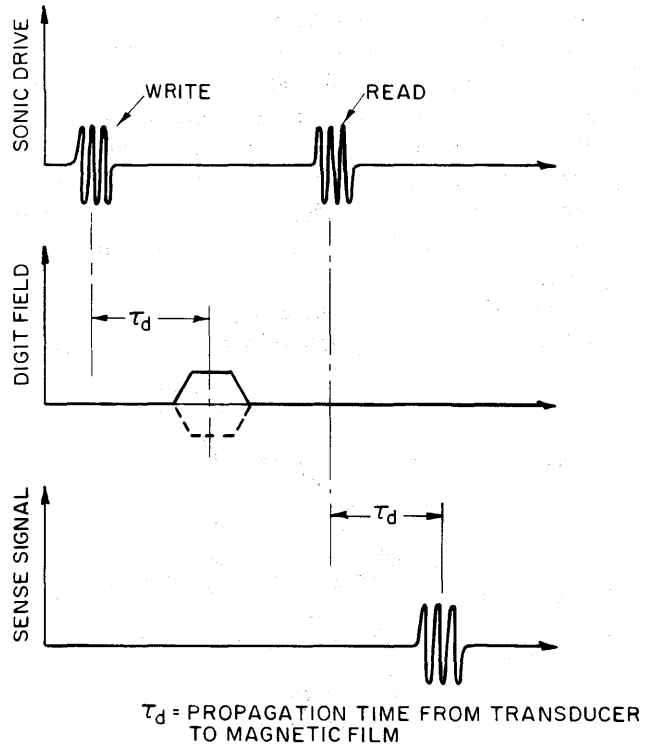


Figure 9. Timing sequence for dynamic testing.

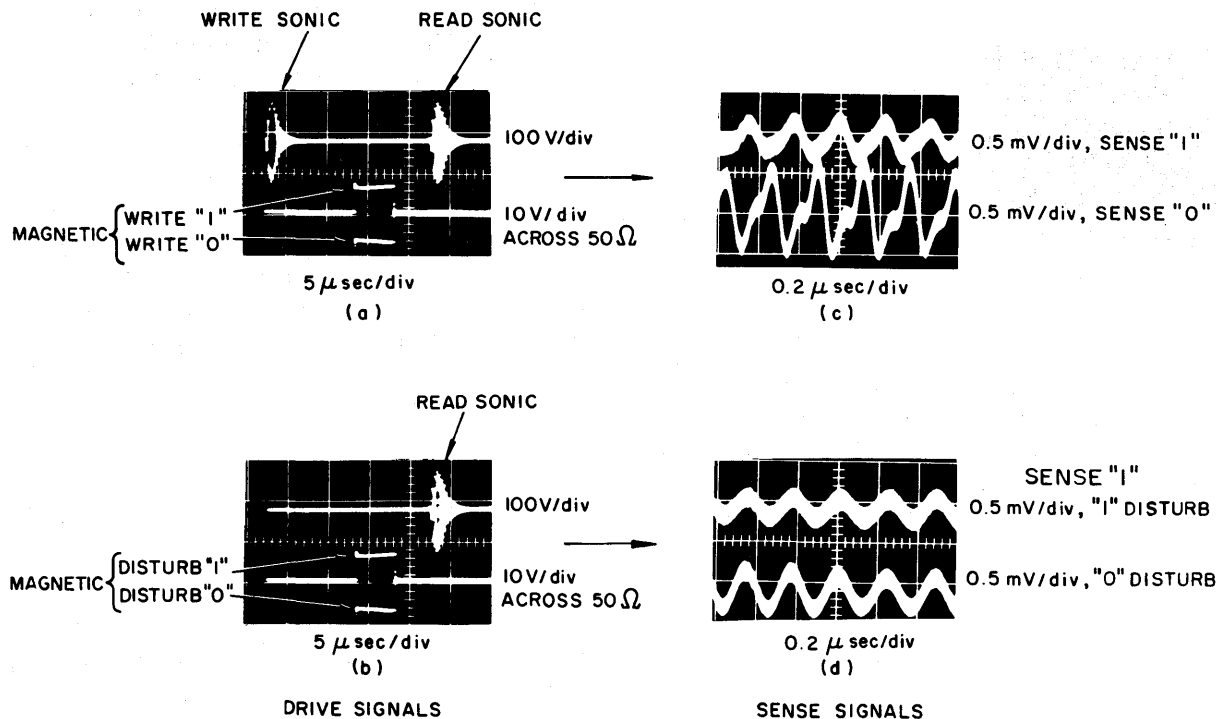


Figure 10. Memory cell operation.

films, direction of the easy axis, and the direction of particle motion in the shear mode wave.

The glass substrate is sandwiched between two copperclad etched circuit boards which form the sense and drive windings for the magnetic films. Unipolar pulses from an external generator are applied to the transducer to generate strain pulses in the substrate. Excitation ranges from 30 to 100 volts and is adjusted in width for optimum transducer response.

Figure 12 shows the sense signal (sense winding parallel to easy axis) generated when a given storage

location is switched into one of the two remanent states. This is accomplished by the coincidence of a strain pulse and a magnetic field pulse applied to the transducer and the drive windings, respectively. As can be seen, the sense signals are bipolar and of an amplitude of 0.5 millivolts.

The magnitude of the sense output may be enhanced by applying a constant hard axis bias during the sense operation. For the device configuration of Fig. 11, a hard axis bias is applied by a pair of Helmholtz coils. The output sense signals are shown in Fig. 13. In the absence of a hard axis bias, the sense signals induced in a winding parallel to the hard axis are negligibly small.

STRAIN WAVE GENERATION

For operations of a high-density sonic film memory, unipolar strain pulses with a width of approximately 10 nanoseconds, and a strain amplitude no greater than 10^{-4} must be radiated into a fused quartz substrate. The most practical method for generating such pulses is by means of piezoelectric transducers.

While a considerable number of materials are piezoelectric, five were selected for investigation.

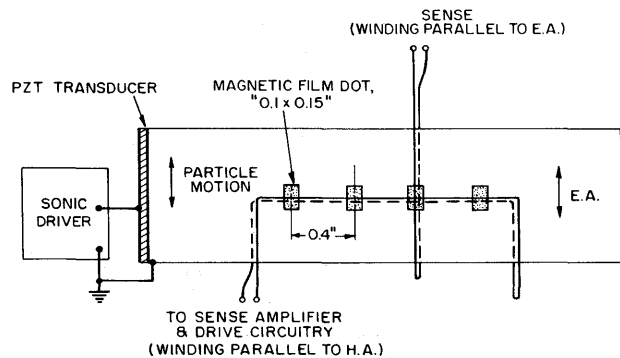
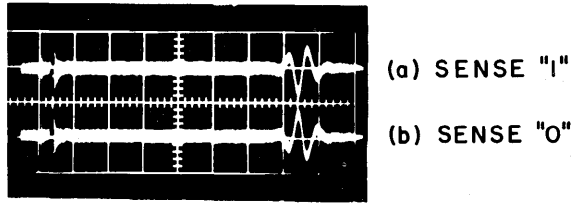


Figure 11. Planar film storage array.



0.5 mV/div., 2 μ sec./div.

Figure 12. Sensing a single bit.

These are quartz, cadmium sulfide (CdS), lithium metagallate (LiGaO_2), antimony sulfur iodide (SbSI), and lead zirconate-lead titanate (PZT). The emphasis was on quartz which is readily available and capable of fundamental mode operation up to approximately 100 MHz. PZT is used in the low MHz range, because of its high conversion efficiency. The three other materials were investigated because of their great potential. A brief survey is given below of the piezoelectric characteristics of these materials.

Material Characteristics

1. *Quartz*. For X-cut quartz, vibrating in its thickness mode, strains generated¹⁰ by an applied electric field, E , is:

$$s = d_{11} \times E$$

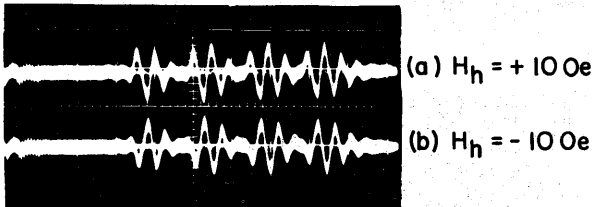
where d_{11} is the transmission sensitivity. For quartz $d_{11} = 2.3 \times 10^{-12}$ coulombs/newton.

The strain propagating in a fused quartz substrate may be evaluated by using a transducer as a receiver. The electric field generated by the transducer due to an incident strain pulse is:

$$E = g_{11} \times T = g_{11} \times Y \times s$$

where T is the incident stress, Y is Young's modulus for fused quartz, and g_{11} is the reception sensitivity. For quartz,

$$g_{11} = 0.058 \text{ volt-meter/newton.}$$



0.5mV/div., 2 μ sec /div.

Figure 13. Sense signal from a 4-element storage array.

For an exact determination of strain at a specific location, losses in the bond and the substrate must be taken into account.

To generate a narrow strain pulse, a wideband transducer is required. The passband of a transducer is defined almost entirely by the ratio of the acoustic impedance of the transducer to that of the delay medium.¹¹ Fortunately, the characteristic acoustic impedance of quartz (15.1×10^{-6} kg/m²-s), and fused quartz (13.1×10^{-6} kg/m²-s), are almost equal. A quartz crystal loaded with fused quartz and air backing has a mechanical quality factor $Q_m \approx \pi/2$.

The objective is to have a strain pulse with a rise or fall time τ of the order of 5 nsec. This requires a transducer bandwidth B of 80 MHz ($\tau \times B \approx 0.4$) and a natural resonance frequency in excess of 60 MHz.

For a 1 mil thick transducer, the resonance frequency is 100 MHz. To generate a strain of 10^{-4} requires 1100 volts. The emf generated by the transducer operating as a receiver due to an incident strain of 10^{-4} is 11.5 volts.

Quartz has a low dielectric constant ($K = 4.5$); the acoustic losses are extremely small and the capacitive loading is unimportant for the required pulse response.

2. *Cadmium Sulfide, (CdS)*. CdS has a higher transmission sensitivity than quartz:

$$d_{33} = 10.3 \times 10^{-12} \text{ c/n}$$

The single crystal material is difficult to lap down to thin wafers (below 3 mils thick). However, CdS can be evaporated and recrystallized¹² giving an attractive approach for making practical high frequency transducers. Fundamental mode operation at 100 MHz requires a film thickness of 17.5μ , and 175 μ will generate strain of 10^{-4} .

The characteristic impedance of CdS is 21.5×10^{-6} kg/m²-s and there is a mismatch to fused quartz representing a 2.3 dB loss, which must be added to bond losses caused by imperfect bonding. With evaporation techniques, however, inherently good bonds are expected.

3. *Lithium Metagallate, (LiGaO₂)*. Relatively little is known about this material.¹³ Single crystals are mechanically hard and have a low dielectric constant. The acoustic Q ($> 10^5$) has been measured. The electromechanical coupling coefficient $k_t = 0.25$ and $d_{33} = 7.7 \times 10^{-12}$ c/n. The compound seems to

have most of the attractive qualities of quartz, but is capable of generating three times higher strain for the same drive voltage. The characteristic impedance is $27 \times 10^6 \text{ kg/m}^2\text{-s}$ and the mismatch of LiGaO₂ to fused quartz causes a loss of 3.5 dB.

4. *Antimony Sulfur Iodide, (SbSI)*. This material has only recently been investigated as a piezoelectric transducer.¹⁴ It is the strongest piezoelectric material known. At 18°C, $d_{33} = 2000 \times 10^{-12} \text{ c/n}$ which is three order of magnitude greater than for quartz. The coupling coefficient k_t ranges from 0.8 to 0.9. Disadvantages are the very high dielectric constant, and the Curie temperature of 22°C.

5. *Lead Zirconate-Lead Titanate, (PZT)*. This ceramic compound is readily available and has been investigated thoroughly. Its transmission sensitivity $d_{33} = 260 \times 10^{-12} \text{ c/n}$. A potential of 110 v across a 20 mils thick transducer produces a strain of 10^{-4} . The relative dielectric constant is large ($K_3 = 1300$), and makes unipolar pulse operation difficult. The characteristic impedance is $30 \times 10^6 \text{ kg/m}^2\text{-s}$ and losses due to mismatch to fused quartz are 4.2 dB.

Experimental Data

1. *Quartz*. Transducers having thicknesses of 1, 2, and 3 mils were tested in the longitudinal mode of vibration. One-mil thickness corresponds to a resonant frequency of 100 MHz, and is about the upper practical limit for fundamental mode operation.

For the delay material we used fused quartz because of its low attenuation characteristics. Most of the material used is commercial grade. The end surface of the fused quartz rod is given a conductive coating of gold.

A good bond between the crystal and the metalized rod is difficult to achieve at high frequencies. The bonding agent finally adopted is waterglass (potassium silicate). The tensile strength of the bond is high and its characteristic impedance is close to quartz ($\approx 9 \times 10^6 \text{ kg/m}^2\text{-s}$), although the bond is not of a permanent nature.

We tested the samples to estimate strain magnitude, determine pulse shape, optimize the width of the drive pulse, determine amplitude/frequency response, signal-to-noise ratio, insertion loss, attenuation in bonds and delay medium, and observe extraneous modes. The transducer assemblies are mostly operated single-ended: the same crystal is used for both transmitting and receiving with the signal reflected from the polished end of the fused quartz.

A figure of merit F may be defined as the ratio of the transducer output voltage when operating as a receiver to the exciting voltage applied to the same transducer. The figure of merit measured for the best samples is:

$$F = 1.4 \times 10^{-3}$$

for a 3 inch long single-ended sample

and

$$F = 2.5 \times 10^{-3}$$

for a 7/8 inch long single-ended sample.

These samples were excited with unipolar pulses from a mercury pulse generator. The output voltage is measured across a 50- Ω termination. Up to 1000 v are applied. The figure of merit values include losses in bonds and in the delay material.

The losses are measured at 100 MHz by observing the exponential decay of reflected echoes while the transducer is excited with bursts of 100 MHz sine waves. Losses in a bond are 2.45dB, and the intrinsic loss in fused quartz is 0.26 dB/cm. Similar values are given in the literature.¹⁵

Figure 14 illustrates pulse response from 1 mil thick crystals. The output amplitude can be maxi-

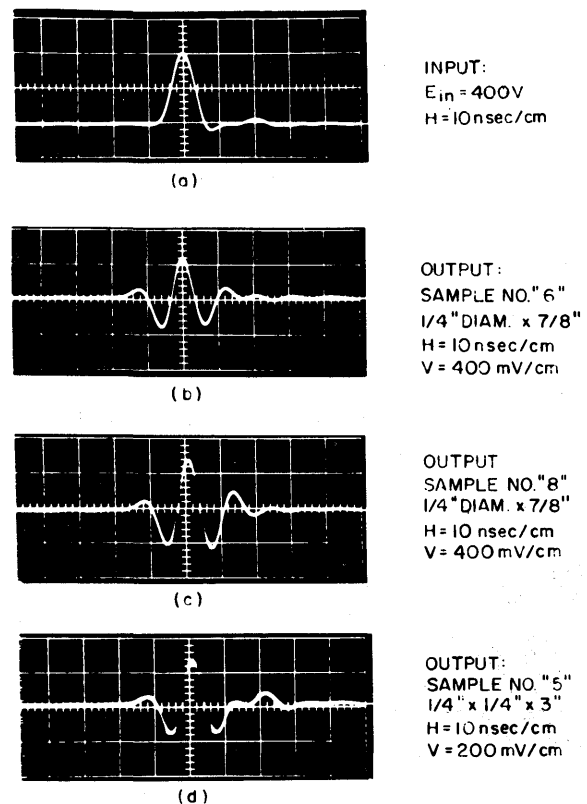


Figure 14. Output from three different samples of quartz transducers, single-ended.

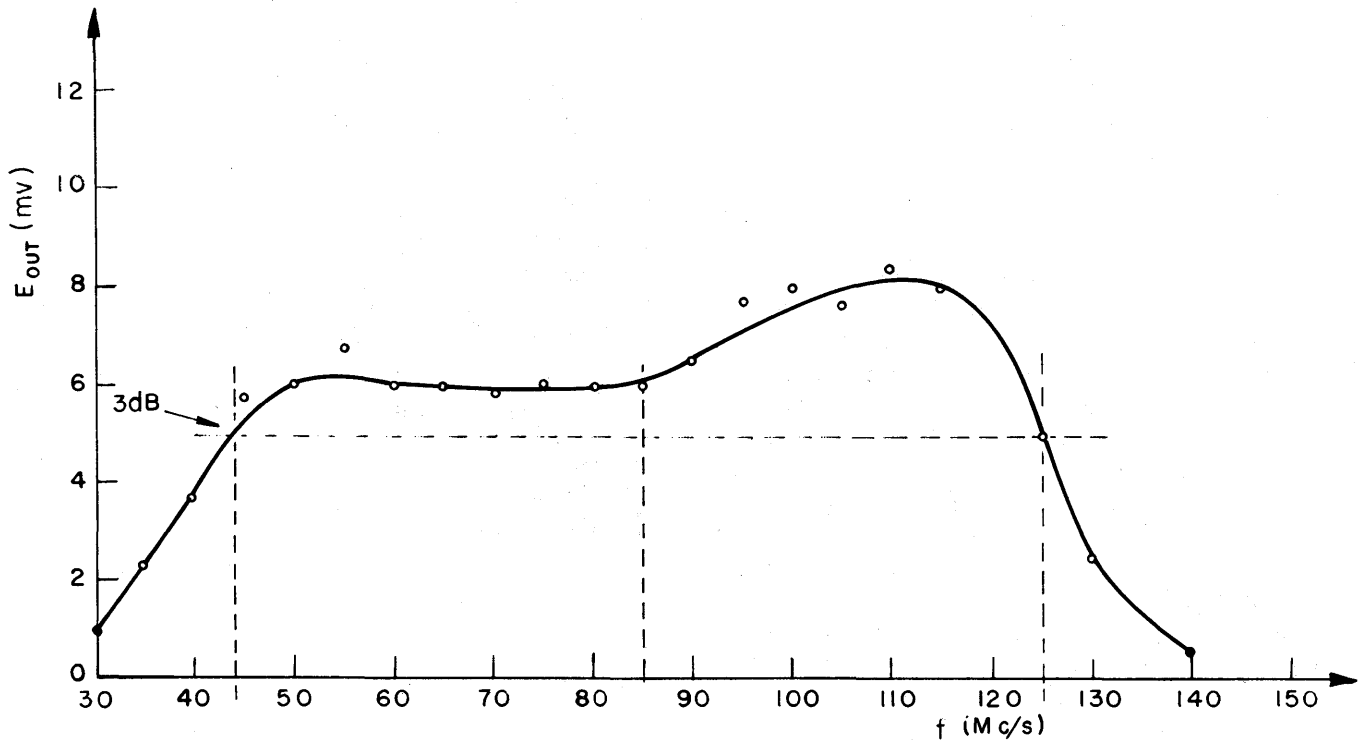


Figure 15. Frequency response characteristic for 1-mil quartz transducer.

mized by adjusting the width of the input pulse. The optimum input pulse width τ_{in} is related to the transducer resonance frequency fr by:

$$\tau_{in} = \frac{1}{2fr} = 5 \text{ nsec}$$

We measured $\tau_{in} = 7 \text{ nsec}$ and the output pulse width is $\tau = 6 \text{ nsec}$.

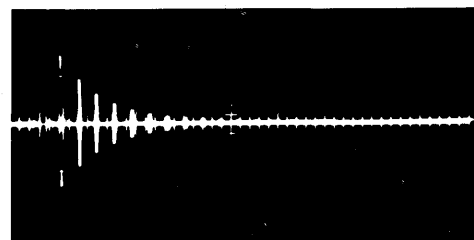
The pulse response is indicative of a wideband transducer. This is confirmed in the amplitude/frequency response curve shown in Fig. 15. The 3-dB bandwidth in this curve is 100%. The signal-to-noise ratio of the sample is 67:1. The S/N ratio is defined as the ratio between the first echo and the largest spurious signal following it. The measurement is done with short bursts of sine waves at 100 MHz. While the transducer ideally should vibrate only in its longitudinal mode, some shear modes, mode conversions, and spurious modes will always be generated.

2. *Cadmium Sulfide, (CdS)*. High-resistivity films which are piezoelectric active in the 100 MHz and lower frequency range have been produced by an approach based on the de Klerk method.¹⁶

Figure 16 shows the response from a 1 inch long sample operated with bursts of sine waves at 100 MHz. The figure of merit (unipolar operation) is $F = 30 \times 10^{-3}$.

3. *LiGaO₂ SbSI, and PZT*. Lithium metagallate has been used at a natural resonance frequency of 19 MHz. The figure of merit is about three to four times higher than observed for quartz.

SbSI is a material difficult to cut and polish be-



H: 20 μ s/cm. V: 200mV/cm.

FUSED QUARTZ ROD: 1" x 0.25" DIA.

Figure 16. Echo response from evaporated CdS transducer ($f = 100 \text{ MHz}$).

cause of its soft porous nature. One 5 mil thick sample operated at 23 MHz. However, the figure of merit is far below theoretical expectation.

Experiments with PZT-4 material were carried out for samples as thin as 5 mils and fundamental operation of 17 MHz. The figure of merit for unipolar pulse operation is 130×10^{-3} . This is two orders of magnitude higher than for quartz.

4. *Guided Wave Propagation.* To check the feasibility of strain wave interaction with anisotropic thin magnetic film a guided wave structure is employed. In this structure, planar films are evaporated on a standard microscope glass slide ($1 \times 3 \times 0.040''$). The glass slide serves as the delay line. As shown in Fig. 17, a shear mode transducer with indicated polarization is attached to the end of the glass slide. The end-surface is first polished and metalized with an evaporated gold electrode. The transducer is bonded to the substrate with silver epoxy.

According to earlier work,¹⁷ an infinite number of width shear modes can propagate in a strip of infinite width. All of these modes are dispersive with cut-off characteristics except for the zero-order mode, which propagates at all frequencies from zero upwards. At frequencies sufficiently low so that an acoustical wavelength is greater than twice the strip thickness, only the zero-order mode can propagate and it is completely free from dispersion. From the viewpoint of acoustical transmission alone, the fundamental nondispersive shear mode of propagation offers the most advantageous characteristics. Also the shear wave velocity is less than the velocity of extensional waves by a factor of about three-fifths, and permits greater bit packing density.

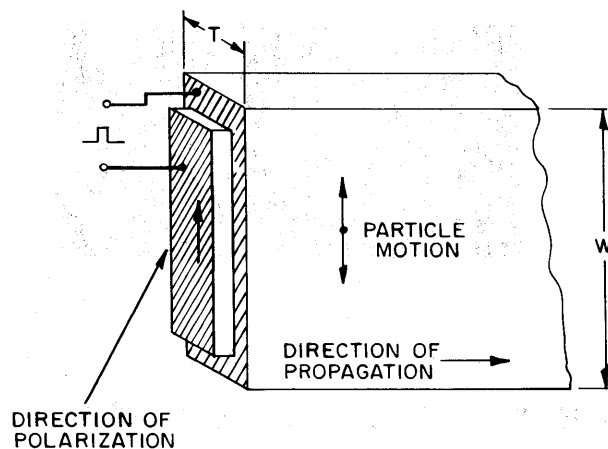


Figure 17. Strip delay line with shear mode transducer.

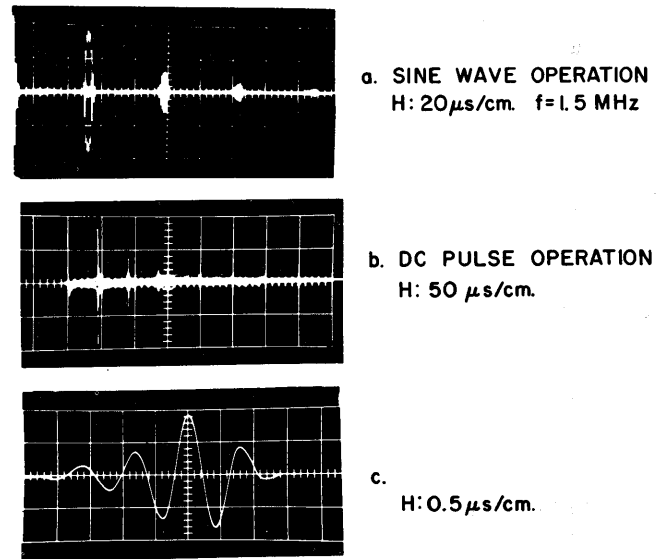


Figure 18. Echo response: PZT shear mode transducer on strip delay line $1 \times 3 \times 0.040''$.

The lowest cut-off frequency of the longest dispersive mode for the chosen substrate geometry is $f_c = 1.9$ MHz. The resonance frequency of the transducer used for testing is chosen 1.5 MHz, and the acoustical wavelength is 100 mils. Since the strip width is many times the wavelength, the main lobe is narrow in the width direction. Furthermore, the minor surfaces are coated with an absorbing material to lower reflections of energy from secondary lobes.

The echo pattern reflected off the end of the 3 inch long strip is shown in Fig. 18. Figure 18a illustrates sine wave drive, while unipolar pulse operation is shown in Fig. 18b. The first pulse echo is expanded in Fig. 18c. The optimum width of the input pulse is 330 nsec, and the bandwidth of the transducer is 34%.

MAGNETIC MATERIALS

Requirements

The objective of the material investigation is to find a composition with high sensitivity to strain and low threshold values $-H_c$ or H_k .

In addition, the films must have the following properties: good orientation of the easy axis, good squareness of the hysteresis loop, and low dispersion.

Nickel-iron magnetostrictive alloys have thresholds below 10 Oe, good squareness, and good orientation. The highest strain sensitivity encountered with these materials yields a change in threshold by a

factor of 2 for a strain of 10^{-4} . For practical operation, a sonic memory material with a similar change for a strain of 10^{-5} is desirable.

Materials Investigation

To find a desirable film material special instrumentation was constructed for production and static testing of samples as described below.

Fabrication Facilities. For this phase the films are produced by evaporation. Figure 19 shows the evaporator used, as well as some of the jiggging especially constructed for this purpose. The melt material is evaporated from an alumina crucible by rf induction heating. Eight substrates can be coated serially in a single run. The vacuum during evaporation is of the order of 10^{-6} torr.

The substrates are heated to 350°C by specially developed individual ceramic heaters. The heat distribution over a slide is rather critical. A thorough investigation shows a maximum variation of only $\pm 5^{\circ}\text{C}$ from slide to slide and over a single slide from its edge to the center. The temperature of the slides is closely monitored by thermocouples.

Thickness is measured by a crystal monitor. The reading is differentiated to produce a rate reading, which in turn is fed back to control the power output of the rf power supply to achieve a constant rate. Automatic shuttering is used to produce films of very uniform thickness.

The films are evaporated and cooled in a uniform magnetic field of 50 Oe. In Fig. 19, the Helmholtz coils used are visible. One of the coils is swung back for access to the jiggging.

Both glass and fused silica are used as substrates. Cleaning of the substrates is accomplished by boiling in chromic acid, ultrasonic shaking and repeated rinsing in acetone, alcohol, and distilled water. Patterns of spots are produced by masking. Figure 20 shows some of the films used in this program. In order to prevent oxidation of some of the more active films, they are coated with plastic spray immediately after opening of the bell jar.

Static Testing Facilities. In order to measure the magnetic parameters of the films under various conditions of static strain, an unique hysteresis loop tester was constructed; the active part of which is shown in Fig. 21.

This instrument permits testing of film-spots of various diameters from $\frac{1}{2}$ inch to 20 mils. To

accomplish this, sensing heads of various sizes are used. The sensing heads consist of coils, wound in a figure-8 pattern, each half consisting of 200 turns of small wire.

Magnetic drive at 10 kHz is applied to the sample with a maximum of 60 Oe peak-to-peak.

Strain can be applied to the sample—both tension and compression. The substrate is cemented with epoxy into bakelite jaws which are coupled to a pneumatic system. The system can apply static strains in excess of 10^{-4} . Strain gages are used to monitor the strain induced in the substrate.

Both the sensing coil and the film sample can be rotated and moved laterally with respect to the driving field. A DC bias field at right angles to the driving field is also provided.

Materials Tested. Figure 22 shows the Ni-Fe-Co melt compositions tested to date. Samples were prepared corresponding to the points indicated in the figure. The films were then tested by fluorescence and wet chemical analysis techniques to determine the exact composition. Note that the chart in Fig. 22 gives the *melt* compositions rather than the film compositions. The exact percentages for the eight films evaporated serially from a melt differed by a few percent both from the melt and among themselves. The films were then subjected to testing on the hysteresis loop tester, and, selectively, to X-ray diffraction analysis to determine the crystal structure.

Results. The best results obtained so far are from melts in region A on the chart of Fig. 22. Films from melts more iron-rich than that end up with a mixture of a Ni-Fe-Co alloy in a body-centered (α) or face-centered (γ) cubic crystal and precipitated α -iron. More cobalt-rich films sometimes give a mixture of body-centered cubic (α) and hexagonal (ϵ) phases; this causes instability and very high values of coercivity. Nickel-rich films have lower magnetostriction, which actually goes through a null along the line indicated in the chart (Fig. 22).

In region A one can obtain very good films, provided a single crystalline phase is achieved. Substrate temperature, surface cleanliness, rate of deposition and other factors determine the film characteristics. Annealing may be used to improve film characteristics.

Figure 23 shows the hysteresis loop of an especially good film obtained from a melt in region A. The photograph shows the hysteresis loop in the direction of the original easy axis under various con-

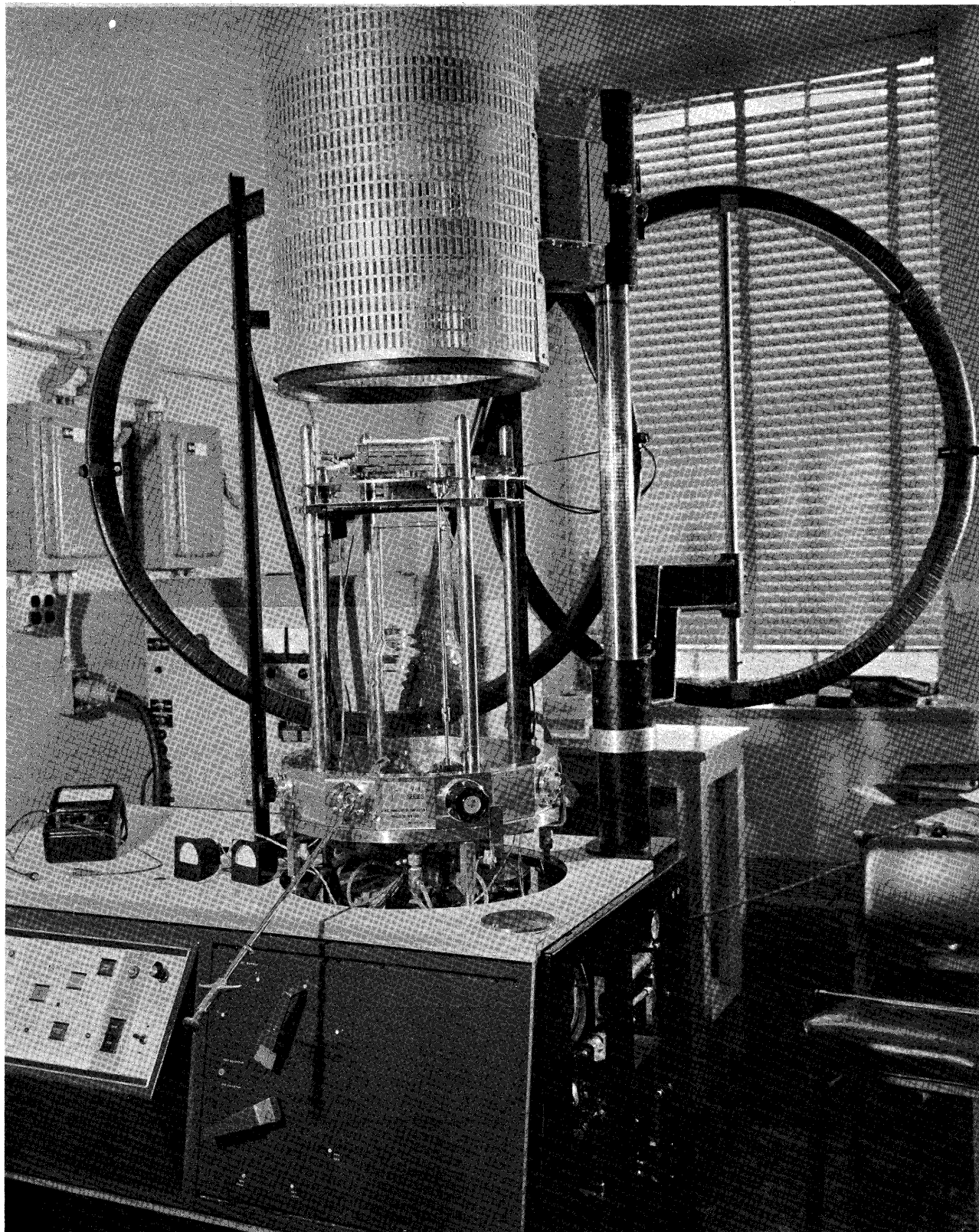


Figure 19. The main evaporator, with bell jar open.

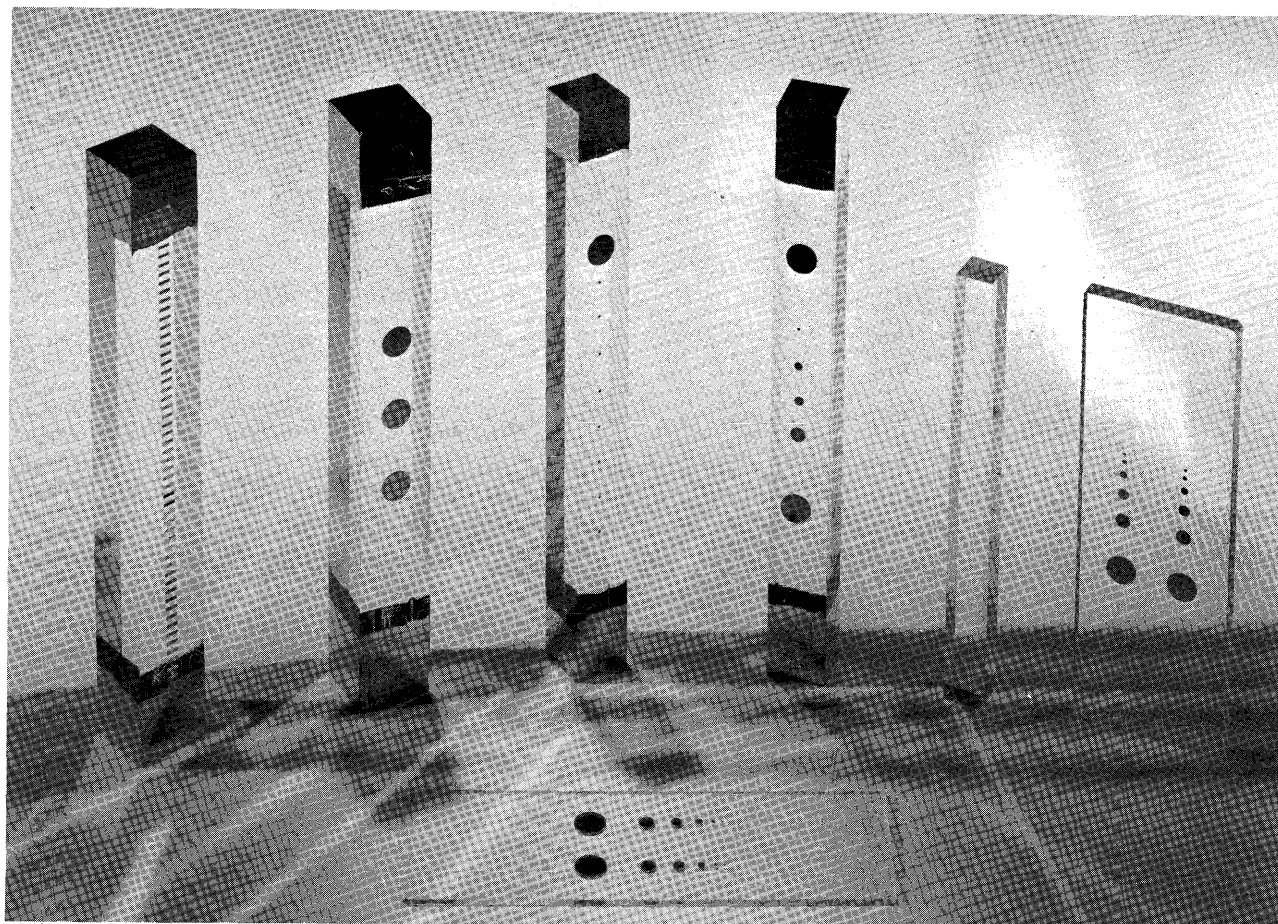


Figure 20. Typical films produced for testing.

ditions of strain, applied along the original "hard" axis direction. The film is well oriented, with a square loop in the "easy" direction; the unstrained $H_c = 7.7$ Oe, $H_k = 3.8$ Oe. Under strain the easy axis rotates 90° for a strain of approximately 4×10^{-5} . A strain of 1.6×10^{-5} decreases the switching threshold in the original easy axis direction by a factor of 2 (see Fig. 24). Some films made from this melt composition were used in the dynamic experiments described in the section on memory cell test data.

CONCLUSIONS

The experimental data presented show conclusively that rotational switching of thin magnetic films with uniaxial anisotropy may be accomplished by the coincidence of strain pulses and magnetic field pulses. The tested films exhibit thresholds for both strain and magnetic fields. Nondestructive read signals are

obtained by subjecting a film element to a strain pulse.

The relative magnitude of the current pulses required to switch a film is compatible with semiconductor circuitry—of the order of 1 ampere. The sense signals derived from elements of 0.1 inch diameter for a strain pulse of 100-nsec risetime are 0.5 millivolts. The magnitude of the sense signal is proportional to the diameter of the film spot and inversely proportional to the risetime of the strain pulse. For smaller film spots, shorter risetimes may be used to maintain the sense signals outputs. Thus for a 5-nsec risetime and a spot size of 5 mils, the sense output is expected to be of the order of 0.5 mv.

Transducer developments showed that evaporated CdS films of the required thickness generated high amplitude strain pulses of the required speed with modest drive voltages. The developed magnetic materials with magnetostriction coefficients considerably higher than permalloy, further reduce the transducer

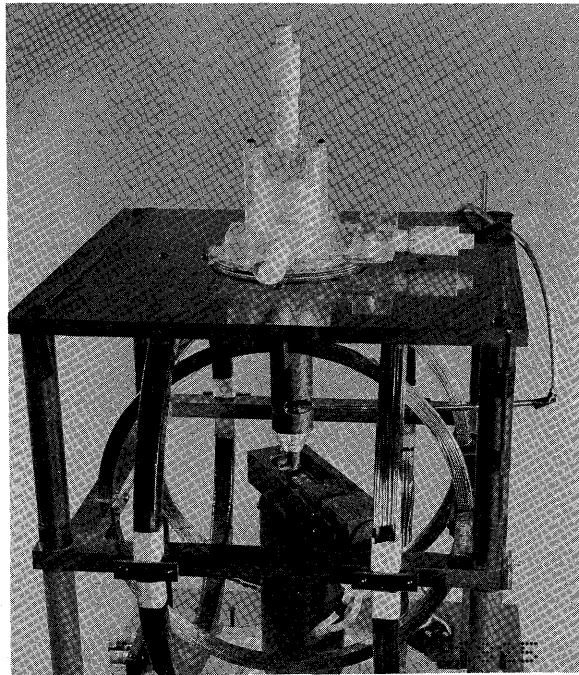


Figure 21. Close-up view of the looper fixture.

drive voltage to levels compatible with semiconductor circuitry.

Magnetic interactions between adjacent bits will undoubtedly be of importance at high bit densities. For random access film memories, bit densities of the order of 50 per inch appear feasible.¹⁸ For the sonic film memory, the number of disturbing magnetic field pulses to which a given bit is subjected does not exceed the number of bits in a block. Thus it is ex-

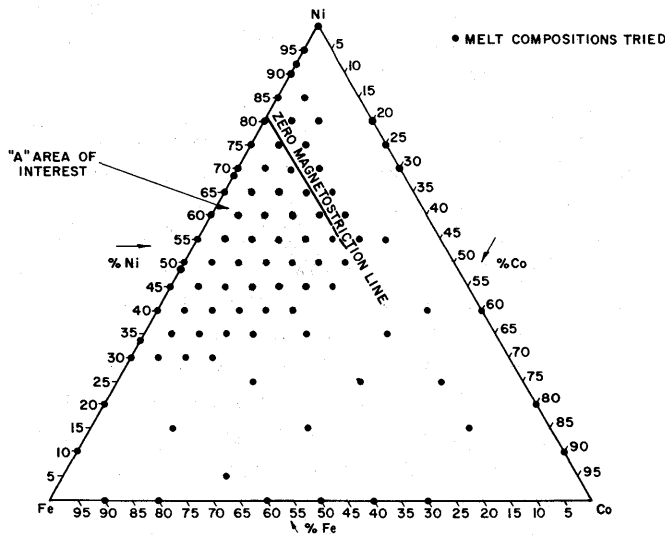


Figure 22. Nickel-iron-cobalt compositions.

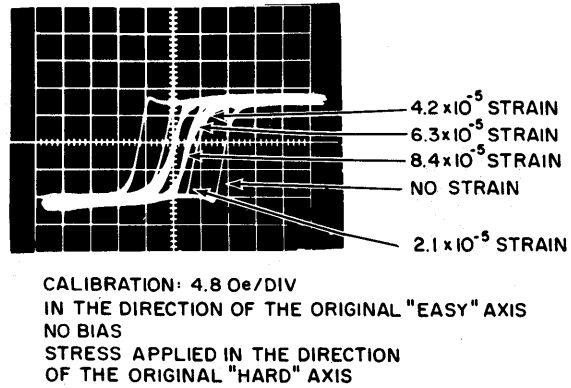


Figure 23. Stress sensitivity of a magnetic film made from a 25% Fe, 15% Co, 60% Ni melt.

pected that disturb effects will be of smaller magnitude allowing high bit densities.

Achieving high bit densities and block lengths of several thousand bits is crucial to the economic success of the sonic film memory. Continuing effort is directed toward achieving this goal.

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions and guidance of Dr. J. A. Rajchman, Director, Computer Research Lab, RCA Research Center, Princeton, N.J.

Mr. S. Hotchkiss contributed in the area of mechanical design and was primarily responsible for the design and construction of the hysteresis loop

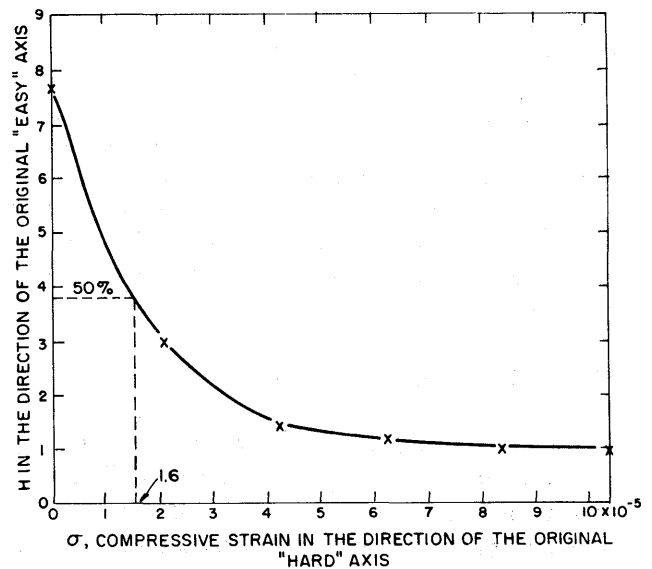


Figure 24. Coercivity in the "easy" direction vs strain.

tester described in the paper. The authors wish to express their appreciation for his help.

Mr. D. Leibowitz developed the technique for fabricating CdS film transducers. His efforts are greatly appreciated by the authors.

Mr. J. Walentine developed an extremely ingenious technique for reliable bonding of quartz transducers to quartz substrates.

Mr. A. Monsen and Mr. T. Ward were instrumental in many phases of this work and were primarily responsible for device and sample preparation. The authors greatly appreciate their help.

Finally, the authors wish to acknowledge the continued interest in the progress of this work by Messrs D. Haratz and D. Hadden of the U.S. Army Electronics Research Laboratory, Fort Monmouth, N.J.

REFERENCES

1. B. H. Gray, D. R. Hadden, Jr., and D. Haratz, "Block Oriented Random Access Memory," Proceedings National Symposium on The Import of Batch Fabrication on Future Computers, Los Angeles, Apr. 1965.
2. J. A. Rajchman, RCA Technical Notes, RCA TN No. 346 (1959).
3. J. W. Gratian and Freytag, "Ultrasonic Approach to Data Storage," *Electronics*, May 4, 1964, pp. 67-72.
4. H. Weinstein, "An Investigation of Serial, Nonvolatile Computer Memories Based on a Magnetoelastic Interaction in Ferromagnetic Storage Media," Ph.D. dissertation, Polytechnic Inst. of Brooklyn, E. E. Dept., 1965.
5. T. R. Long, "Electrodeposited Memory Elements for a Nondestructive Memory," *J. Appl. Phys.*, vol. 31, p. 123s (1960).
6. T. S. Crowther, "Angular and Magnitude Dispersion of the Anisotropy in Magnetic Films," *J. Appl. Phys.*, vol. 34, p. 580 (1963).
7. E. N. Mitchell, G. I. Lykken, and G. D. Babcock, "Compositional and Angular Dependence of the Magnetostriction of Thin Iron-Nickel Films," *J. Appl. Phys.*, vol. 34, p. 715 (1963).
8. H. L. Pinch and A. A. Pinto, "Stress Effects in Evaporated Permalloy Films," *J. Appl. Phys.*, vol. 35, p. 828, (1964).
9. H. Weinstein, "Static and Dynamic Stress Effects in Cylindrical Ferromagnetic Films," 11th Annual Conference on Magnetism and Magnetic Materials, San Francisco, Nov., 1965.
10. W. P. Mason, *Physical Acoustics and the Properties of Solids*, Van Nostrand, Princeton, N.J., 1958.
11. Brockelsby et al, *Ultrasonic Delay Lines*, Iliffe Books, London, 1963.
12. N. E. Foster, "Ultra-High Frequency Cadmium-Sulfide Transducers," *IEEE Transactions*, SU-11, Nov. 1964.
13. J. P. Remeika and A. A. Ballman, "Flux Growth, Czochralski Growth, and Hydrothermal Synthesis of Lithium Metagallate Single Crystals," *Appl. Phys. Letters* vol. 5, no. 9 (1964).
14. D. Berlincourt, et al, "The Piezoelectric Effect in the Ferroelectric Range in SbSI," *Appl. Phys. Letters*, Feb. 1964.
15. W. P. Mason, *Physical Acoustics*, Academic Press, New York, 1963, vol. I, part A.
16. J. de Klerk and E. F. Kelly, "Vapor-Deposited Thin-Film Piezoelectric Transducers," *Rev. Sci. Instr.*, Apr. 1965.
17. A. H. Meitzler, "Ultrasonic Delay Lines Using Shear Modes in Strips," *IRE Transactions on Ultrasonic Eng.*, 1960.
18. Q. W. Simkins, "A High Speed Thin-Film Memory—Its Design and Development," *Proceedings 1965 Fall Joint Computer Conference*, vol. 27, part 1, supplement.

ENGLISH FOR THE COMPUTER

Frederick B. Thompson

*California Institute of Technology
Pasadena, California*

What about English as a programming language? Few would question that this is a desirable goal. On the other hand, I dare say every one of us has rather deep reservations both about its feasibility and about a number of problems that it entails.¹ This paper presents a point of view which gives some clarity to the relationship between English and programming languages. This point of view has found substance in an experimental system called DEACON. The second paper in this session will describe the specific DEACON system and its capabilities.

There is one source of these reservations that we should recognize, and that is the fact that we have no adequate notion of the nature of natural language and no precise description of its vagaries. It is for this reason that most of those working on language problems have concentrated on programming languages or confined themselves to syntax. However, the semantics of natural language pose important problems. These remarks are related to those problems.

The excellent work that has been done on programming languages, in particular on syntax-directed compiling and its associated semantics, and work in the area of symbolic logic have cast much light on the natural language problem as well. It has illuminated some very real difficulties. It has also illuminated some aspects which can be exploited to good ends. And more important, it has allowed the sepa-

ration of the deep difficulties of dealing with natural language from some of these exploitative opportunities. We shall build upon this work as well as on recent work in linguistics.

What can be said about English as a computer language? There are certain aspects of a very difficult nature that are involved in a full-blown natural language, namely the fact that it is self-referent. In English, we can speak of English; we are doing so at this very minute. We can say such things as: "John believes Mary lies." Worse, we can say: "This sentence lies." And we are all aware of the implications of this fact as discussed by Tarski,² Gödel, and Turing.³ When we think of the possible uses of English as a computer language, we realize that little will be lost if we abandon those parts of English which are self-referent or involve indirect discourse. Let us do so.

Much of the world's knowledge is in written form. Computers are being applied to the processing of such documentary information and are being programmed to do some of the more routine functions of the research librarian. To this end they must indeed have a certain understanding of English. Good work is being done in this direction, for example, the work of Robert Simmons.^{4, 5} However, the use of computers to intelligently service documentary material is quite distinct from the use of English as a computer language.

Thus I would like to focus our attention on the use of English to:

1. input information into a computer,
2. instruct the computer to process the information that it has stored away, and
3. query the computer concerning the information it has stored away, and which results from processing.

These are the functions that programming languages perform.

Usually when we think of English, we are tempted to include the traditional patterns of syntactic analysis. But the parts of speech—noun, adjective, verb—are not a part of English, but rather a method that grammarians imposed long ago in their attempts to understand the regularities of structure that are apparent in language. Modern linguists, in their study of syntax, have ramified, redefined, and modified these traditional categories, well aware that they are, at best, an imperfect tool for understanding the structure of language. We shall feel free, therefore, to choose our syntactic categories in whatever way is useful in our analysis, and shall feel no compunction to stick to the traditional parts of speech.

In programming languages, we also find syntactic categories: operator, label, subscripted integer variable, etc. In the formal expression of the syntax of a programming language these categories are used in a fashion parallel to the use of parts of speech in a phrase structure grammar for English.

Typical phrase structure rule for English:

$\langle \text{Verb phrase} \rangle ::= \langle \text{Verb} \rangle \langle \text{Noun phrase} \rangle$

Typical phrase structure rule for a programming language:

$\langle \text{real expression} \rangle ::= \langle \text{add op} \rangle \langle \text{real factor} \rangle$

However, there is a striking difference. The parts of speech of traditional linguistics do not have semantic implications beyond that made explicit by the rules of grammar. They can be fully characterized as non-terminal symbols which are used in expressing the recursive relationships of English structure. In times past, loose attempts to define these parts of speech in terms of meaning have been made. However, since Bloomfield such attempts have fallen into disrepute.

In sharp contrast, the syntactic categories used in the description of programming languages have clear

semantic implications. An integer variable and a real variable designate two quite distinct entities, independent of how these variables are used syntactically in program statements. In FORTRAN, for example, to say that an expression is a doubly subscripted variable implies a good deal about the associated material in memory, namely that it is a two-dimensional array stored column after column contiguously. The part of speech of a word used in a programming language carries clear structural implications for the way the corresponding material is stored in memory.

The work of Irons,⁶ and of those that have followed him in the development of syntax-directed compiling, has exploited this relationship and indeed has gone much further. With each rule of grammar there is associated a corresponding segment of code which expresses the operations on memory structures implied by a grammatical phrase to which the rule applies. The syntactic analysis of a program statement in terms of these rules of grammar provides the necessary directions for compiling these segments of code into a computer program which expresses the semantic context of the statement. One of the most elegant formulations of this point of view has been given by Wirth and Weber in their paper: "EULER: A Generalization of ALGOL, and its Formal Definition."⁷

The following definitions of a formal language are a straightforward generalization of these developments, for example, of the definitions given in the Wirth-Weber paper.

A *syntax* is an ordered quadruple (V, Φ, B, s) where V is a vocabulary; Φ is a finite set of syntactic rules ϕ_i (these rules may be assumed to be of the form $x \rightarrow y$, where x and y are strings from V); B designates the terminal symbols, a subset of V ; and s is an element of $V - B$ (which can be thought of as the part of speech "sentence").

The rule $x \rightarrow y$ is to be read "the substring x may be rewritten as y ." Thus it permits a string $w x z$ to be rewritten as $w y z$. If a string u can be transformed into a string v by successive rewritings of substrings according to the syntax rules, then u is said to produce v ; in symbols, $u \xrightarrow{*} v$. In a derivation such as $u \xrightarrow{*} v$, a sequence $(\phi_1, \phi_2, \dots, \phi_m)$ of

syntax rules is applied. The inverted sequence $(\phi_m, \phi_{m-1}, \dots, \phi_1)$ is called a parse of v from u . A string x is a sentence if $s \xrightarrow{*} x$, and all of its symbols are in B , i.e., are terminal symbols.

If all the rules are of the form $x \rightarrow y$, and x is always a single element of V , the syntax is called a context-free phrase structure grammar. Although context-free phrase structure grammars are convenient to work with, it is known that they are not adequate to describe current programming languages, nor do they appear at all adequate for description of natural language. On the other hand, it is known that any language whose sentences are recursively enumerable has a syntax as defined above, i.e., has a Post production grammar⁸; thus our definition is as general as one would desire. In practice, one may wish a more complex form of syntactic rule—one that specifies more completely the character of the strings x for which a substitution may be made. Such rules will be discussed at length below.

The terminal symbols B can be divided into two parts: $B = F \cup R$. R , the referent symbols, are those which refer to specific values. Typically, variables and constants are referent or English words such as "house" and "red." F , the function symbols, are exemplified by delimiters or by the English words "and" and "all." They play a quite different role from referent words, as will be seen below.

The referent words of a language are differentiated by the type of objects they may denote. In programming languages, referent symbols include integer variable, real two-dimensional array variable, list name, function name, etc. Moreover, phrases (derivations from referent symbols) may also be differentiated by the types of objects they denote. Thus in FORTRAN, not only do J and I denote integers, but so also does $J + I$; in LISP, not only are A and B list names but so is $(A B)$. When we examine the rules of syntax for a programming language, we find that the nonterminal symbols appearing in these rules are names for these categories of objects which the corresponding referent symbols or phrases may denote. They may also contain certain syntactic information (for example, the difference between a term and a factor), but there is indeed a relationship which relates each nonterminal symbol to a category or group of categories of objects which the referent symbols and phrases may denote. These categories are the environment for the language.

An *environment* E is a finite set (C_1, C_2, \dots, C_n) of categories of memory structures. The C_i need not be disjoint.

For example, the environment for FORTRAN is the set of integer and floating point scalars, one-, two-, and three-dimensional arrays, and Boolean elements.

An *interpretation rule* ψ defines an action (or sequence of actions) involving the objects of an environment E . This formalizes a semantic counterpart of syntax.

A *formal language* L is a septuple $(V, \Phi, F, R, s, \Psi, E)$, where

- a. $(V, \Phi, F \cup R, s)$ is a syntax, and
- b. E is an environment.
- c. There is a correspondence (possibly many-many) between the symbols of $V - (F \cup R)$ and the categories of E .
- d. There is a correspondence between R and objects of the categories of E , thus establishing the initial values of referent symbols.
- e. Ψ is a set of interpretation rules such that a one-one mapping exists between elements of Ψ and Φ , and E is the environment for elements of Ψ .

To be complete, somewhat more than this must be said about the relationship between syntax rules and interpretation rules. We illustrate this with an example:

Rule: $\phi: a_1 a_2 a_3 \rightarrow b_1 b_2 b_3 b_4$

Suppose: b_i corresponds to $C_i \in E$

a_i corresponds to $C'_i \in E$

ϕ corresponds to the interpretation rule ψ .

Then ψ is on $C_1 \times C_2 \times C_3 \times C_4$ to $C'_1 \times C'_2 \times C'_3$. In this example, we have assumed neither side of the rule ϕ contains function words. Function words, being nonreferent, do not enter into the determination of the arguments or values of ψ .

We are now in a position to define the meaning of a sentence of L . The *meaning* $M(x)$ of a sentence x of L is the effect of the execution of the sequence of interpretation rules $\psi_1, \psi_2, \dots, \psi_m$ on the environment E , where ϕ_1, \dots, ϕ_m is a parse of the sentence x into the symbol s , and ψ_i corresponds to ϕ_i for all i .

I should like to rephrase certain of these notions in diagrammatic form to make clear certain of their interrelationships. Let the environment E consist of the categories C_1, C_2, \dots, C_k . Then the relationship between a referent word or phrase x and its value X can be shown by the following commuting diagram, where $p \in V - (F \cup R)$ is the part of speech of x and $C \in E$ is the category associated with p .

$$\begin{array}{ccc} x & \rightarrow & p \\ \downarrow & & \downarrow \\ X & \xrightarrow{\varepsilon} & C \end{array}$$

Consider now the case of a context-free phrase structure rule of grammar $\phi: q \rightarrow p_1 p_2 \dots p_n$. Suppose we have a string $x_1 \dots x_n$ where each x_i is a string of terminal symbols and has previously been parsed to p_i , i.e., $p_i \xrightarrow{*} x_i$. According to the above definitions, there is an interpretation rule ψ corresponding to ϕ such that the following diagram commutes.

$$\begin{array}{ccc} x_1 \ x_2 \ \dots \ x_n & \xrightarrow{\phi: q \rightarrow p_1 \ p_2 \ \dots \ p_n} & \\ \downarrow \ \downarrow \ \dots \ \downarrow & & \downarrow \ \downarrow \ \dots \ \downarrow \\ \psi(X_1, X_2, \dots, X_n) = Y & \rightarrow & (\psi: C \leftarrow C_1 \times C_2 \times \dots \times C_n) \end{array}$$

where X_i is the value denoted by x_i and X_i , as a memory structure (such as an array or list), is in the category C_i . The top half of the diagram shows that the string $x_1 \dots x_n$ can be further parsed by ϕ , i.e., $q \xrightarrow{*} x_1 \dots x_n$. Correspondingly, the value denoted by $x_1 \dots x_n$ is $Y = \psi(X_1, \dots, X_n)$. The interpretation rule ψ is shown as a functor that maps the Cartesian product of the categories C_1, \dots, C_n into the category C .

A more general diagram for a noncontext-free, Post production rule is shown as follows:

$$\begin{array}{ccc} x_1 \ x_2 \ \dots \ x_n & \xrightarrow{\phi: q_1 \ \dots \ q_m \rightarrow p_1 \ \dots \ p_n} & \\ \downarrow \ \downarrow \ \dots \ \downarrow & & \downarrow \ \downarrow \ \dots \ \downarrow \\ \psi(X_1, X_2, \dots, X_n) = (Y_1, \dots, Y_m) & \rightarrow & (\psi: C' \leftarrow C_1 \times \dots \times C_n) \end{array}$$

What conditions must be placed on the interpretation rule ψ ? Considering the matter from the point of view of syntax-directed compiling, it certainly must be the case that the definition of ψ is independent of the particular values X_i and depends solely on the character of the categories C_i of memory structures to which it applies. For example, the code compiled for an arithmetic expression $I + J$, where I and J are integer variables, depends only on this fact that they are integer variables and not upon their particular values. The ψ must be defined

in terms of the structural aspects of the categories alone. Further, ψ should be constructive, i.e., there should be an algorithm for computing $\psi(X_1, X_2, \dots, X_n)$ whenever X_1, X_2, \dots are in the appropriate categories. A general definition of "interpretation rule" can be given satisfying these two requirements, along either the programming line following McCarthy⁹ or constructive set theory following Gödel¹⁰; the details however would take us too far afield here. We shall simply speak of an interpretation rule ψ as being structural and constructive.

Now we come to the point of the matter. The above two diagrams show that the domain of definition of ψ is the whole of the Cartesian product $C_1 \times C_2 \times \dots \times C_n$. There is no particular need for this stringent a requirement. Its domain of definition may be some appropriate subset $K \subseteq C_1 \times C_2 \times \dots \times C_n$. However, just as ψ itself must be defined in terms of the structural aspects of the C_i alone, so also must this subset be identified by restrictions of a similar character. A particular important class of such restrictions are those which refer not only to the parts of speech p_i and their associated categories C_i but also to the existence of certain parsings of the strings x_i and the categories associated therewith. Such rules are of particular importance because the restriction on their domain of application can be stated in terms of parsings of the constituent x_i strings and thus stated in purely syntactic terms. Such rules of grammar for natural languages have been identified by Chomsky and Harris who have correctly stressed their importance.¹¹⁻¹³ These are the transformation rules. The importance Chomsky gives to the concomitant transformation of the

phrase marker (roughly: parsing tree), as well as his condition of the substitutability of strings in elementary transformations, can be seen in the above terminology to insure that the restriction on the domain of ψ is indeed dependent only on questions concerning categories and not on particular values involved (see in particular pp. 300-3 of Ref. 12). Such a condition, we have seen, is exactly the one necessary to insure compilable code in a syntax directed compiler.

An example at this point may be in order. Consider the situation where one wishes to analyze the sentence "John saw Mary and Joan" into the two sentences: "John saw Mary. John saw Joan." Notice that the rule: $NVN.NVN \rightarrow NVN \text{ and } N$. is not adequate, for it does not signal the condition that the first and fifth constituents (namely "John," and "John" in the example sentence) must be identical. This extra condition cannot be simply stated in phrase structure form but is easily and correctly stated as a transformational rule. The passive transformation, $N_1VN_2 \rightarrow N_2 \text{ aux } V \text{ by } N_1$, is another example where an extra condition is necessary to correctly identify the switched positions of subject and object. Indeed, in the formation of many transformation rules, it is desirable that the rule be applied to the entire sentence where the restriction of the domain of the transformation is stated in terms of an analysis of the structure of the sentence. In this case, the phrase structure aspect of the rule takes on the trivial form $s \rightarrow s$ (Ref. 12, pp. 300-303). It is interesting to conjecture that the use of such rules in defining programming languages might well permit the statement of rules covering parentheses conventions in arithmetic expressions without the introduction of superfluous parts of speech as is now done. The compiling time such complex rules entail would, of course, not warrant the change.

The final diagram, encompassing transformational rules, can thus be shown:

$$\begin{array}{ccc}
 x_1 & x_2 & \dots & x_n & \longrightarrow & (\phi: q_1 \dots q_m \rightarrow p_1 \dots p_n) \\
 \downarrow & \downarrow & & \downarrow & & \text{with side condition} \\
 \psi(X_1, X_2, \dots, X_n) = (Y_1, \dots, Y_m) & \longrightarrow & (\psi: C'_1 \times \dots \times C'_m \leftarrow K \subseteq C_1 \times \dots \times C_n)
 \end{array}$$

The explanatory power of the approach presented here can be seen to greater advantage by starting with the semantic aspects rather than the syntax. Let us focus our attention for a moment on the memory of the computer. Considering it independently from any particular program or programming language, it is difficult to say whether it contains any fixed point variables, arrays or list structures. But it unmistakably has a complex, interknotted web of structure. Now consider a structural, constructive interpretation rule ψ , say taking n arguments (where each of its arguments may be considered as an address in memory). We note that the value $\psi(X_1, X_2, \dots, X_n)$ obtained by application of the rule ψ depends on the structure of memory "local" to X_1, X_2, \dots, X_n . This statement follows from the

structural and constructive nature of ψ . In fact, using the particular definition of ψ , we can characterize certain structural categories C_1, C_2, \dots, C_n . If $X_i \in C_i, i = 1, \dots, n$, that is if the structures in memory which can locally be reached from the X_i have the determined characteristics, then we can determine from the definition of ψ precisely what the value $\psi(X_1, \dots, X_n)$ will be, independent of the rest of memory. (On arguments which do not have these structural characteristics, i.e., $X_i \in C_i$ is not true, we can not predict what ψ will do; thus if a program applies a list processing operation to a "non-list" address, the resulting indeterminacy is characterized as a bug).

Suppose we start with a finite number of interpretation rules $\psi_1, \psi_2, \dots, \psi_m$. From there we can determine as above a finite number of structural categories C_1, C_2, \dots, C_k such that the domains of the ψ_i are subdirect products of the C_j 's, that is, the domain of ψ_i is not only a subset of $C_{i1} \times C_{i2} \times \dots \times C_{in_i}$, but can be characterized structurally in terms of relationships among elements identified in the definitions of the C_{ij} 's.

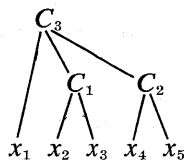
Suppose further that certain arguments X_1, X_2, \dots, X_f are initially given. We now ask what arguments can be reached from the X_i by application and composition of the functions ψ_j ? What is computable? We shall answer this question by relating interpretation rules, categories and initial arguments to our previous discussion.

Correspond to each of the X_i a referent word, or formative; this will be our referent vocabulary R . The categories C_i will constitute the vocabulary $V - (R \cup F)$. If the domain of a rule ψ_i is a subdirect product of $C_{i1} \times \dots \times C_{in_i}$ we will adopt a transformational rule of grammar establishing " $C_{i1}C_{i2} \dots C_{in_i}$ " as a grammatical phrase subject to the structural side condition.

Composition of interpretation rules applied to appropriate arguments can now be seen to have as an exact counterpart the parsing of the corresponding string of formatives. For example (in the case of context-free rules where it is easiest to see):

$$\psi_1(X_1, \psi_2(X_2, X_3), \psi_3(X_4, X_5))$$

corresponds to the parsing tree



where x_i is the referent word corresponding to X_i . Thus those arguments in memory which can be reached starting with the X_i by using functional composition of the interpretation rules are exactly those which can be defined in the corresponding formal language.

It is the underlying structural, constructive interpretation rules on memory which are at the heart of language. From these, the rest including syntax can all be reconstructed. The expressiveness of a formal language reduces to what can be reached from the references of its words by functional composition of its interpretation rules.

Before going on, let us pause to consider the role of function words. According to the definition of meaning given above, a sentence may have multiple meanings, i.e., be semantically ambiguous. This may arise when a sentence has two parsings (though this by itself does not necessarily imply semantic ambiguity). A typical case of ambiguity would occur if parentheses were dropped from all arithmetical expressions. Consider, for example, the expression $I + J \times K$. By convention we assume the multiplication is to precede the addition. If the addition were to be done first, delimiters would be inserted: $(I + J) \times K$. It has been shown by David Benson that any syntax can be made unambiguous through appropriate augmentation by function words, and this in such a way that no possible meaning (in the above sense) will be lost. Thus function words are seen as a device for reducing or eliminating syntactic ambiguity. English sentences are replete with function words, including all sorts of suffixes, prefixes and auxiliary words. Many words play dual roles in this regard, both as pointers which help to establish meaning, and as delimiters; for example, prepositions and determiners.

The above definition of a formal language has been developed in such a way as to show its clear relationship to the notions of syntax directed compilers and programming languages, and to current investigations of the syntax of natural languages. An equally close relationship exists between this

definition and the formal languages of symbolic logic. Rather than formally show this correspondence here, let us see whether we can use the above mechanism to identify the "logic" of a programming language.

The semantic studies which lie at the root of modern logic and metamathematics are based upon an adequate definition of the notion of truth. The fundamental problem can be stated as the problem of determining for a sentence those environments where it is satisfied. To this end, let us choose s , the preferred symbol of our formal language, to be a Boolean variable. In this case, we see that for any sentence x , the meaning $M(x)$ of x will be either "true" or "false." The interpretation rules become the counterpart of Tarski's definition of satisfaction for languages of symbolic logic.¹⁴ A sentence x is logically true, or a tautology, if $M(x)$ is "true" for every initial assignment of values to the referent symbols in R . By this simple means, the notions and results of mathematical semantics can be extended to the generalized notion of formal language given by the above definition.

But what about English? Recall that our interest in this paper is English as a programming language. If we are to develop a syntax-directed interpreter for English, we must first determine what structural categories are to make up its environment E . This question is in some sense a priori to the question of English, for English presumably does not prejudice the structural relationships that exist among the elements of a universe of discourse. On the other hand, the decision as to the memory structures the computer is to use in storing its data is a crucial one. The efficiencies of a programming language depend strongly on policies concerning memory management and structuring. If the universe of discourse is weakly structured with few cross-relationships one would expect any language, English or not, dealing with such subject matter to be inefficient to use and of very limited expressiveness.

The first major issue, then, in using English as a programming language is the same as that for any other programming language, the policy concerning memory management and structuring. When using English, we take for granted a richly connected web of implicit relationships, which we must now make explicit in computer memory. In the current DEACON work, data is organized into ring structures. These structures are similar in many respects to the plex structures defined by Ross¹⁵ and used by

Sutherland in Sketchpad,¹⁶ and are an extension of the notion of list structure.

Once the structural categories of the environment have been chosen, the central issue can be immediately clarified. Each of the referent words and phrases of the language have, as their denotational values, elements which are members of these categories. These categories correspond therefore to parts of speech. Can a syntax for English be developed, using these new parts of speech, which accounts for all of its richness of grammatical structure? A second way to put the same question is this: if the subject matter of English is limited to material whose interrelationships are specifiable in a limited number of precisely structured categories, does English essentially become a formal language as defined above? I believe that the DEACON work to date constitutes a confirmation of this hypothesis.

DEACON makes use of transformational rules as discussed above. It does this in a rather clear way by dividing the syntactic aspect of the grammar rule into two parts. The first is a straightforward phrase structure rule (not necessarily context-free). The second part can be considered as essentially determining whether the constituents fall within the subspace on which the interpretation rule is defined.

In their discussions of transformational grammars, both Harris and Chomsky have pointed out that it is possible through transformations to reduce a complex sentence to a series of interrelated sentences of simple type. Quite independently, we found it most expeditious to use what we refer to as a Verb Table for analysis of a sentence in the DEACON System. The columns in this table correspond quite directly to kernel sentences. The various columns are cross-linked from right to left showing the role of one kernel sentence in defining a constituent of a prior one in the table. The Verb Table is a rudimentary realization of the notion of the deep structure of a sentence. It is interesting to note that the Baseball English language query system by Green et al¹⁷ produces a spec list as an intervening table between syntactic and semantic analysis, which can also be viewed as a realization of the notion of deep structure when applied to the segment of English used in that system.

It is the central thesis of this paper that, when the subject matter of English is limited to material whose interrelationships are specifiable in a limited number of precisely structured categories, English essentially becomes a formal language as defined above. This

hypothesis has far-reaching consequences. It implies that the complexities of natural language arise neither from vagaries of syntax nor from the variety of its subject matter, but rather from the immense complexities of the intervening memory structures which mediate between stimulus and verbal response. The words of the language are keys to the specific structures in memory which carry the referenced information. The relationships established among a particular set of words by a particular sentence are keys to the structural transformations, the interpretation rules, that develop the meaning of the sentence from the structures keyed to its constituent words. If we artificially limit these structural forms, English reduces homomorphically to a formal language.

I should like to make a few remarks on certain issues concerned with the efficacy of English as a programming language. First, it can be said that certain other existent programming languages are English-like in their sentence formats, for example COBOL. What is the essential difference between such languages and extended versions of DEACON? COBOL and other similar languages have chosen a restricted set of formats for their statements which are, to be sure, English-like. However the number of such phrase formats is very limited and any divergence from these formats is excluded. In developing these languages there appears to have been a hesitancy to allow the great plurality of forms which one finds in everyday English, possibly because of a fear that unacceptable levels of ambiguity might arise, possibly because of the acknowledged computing time required by a more elaborate parsing algorithm. In particular little has been done to capitalize on the rich variety of function words which one finds in English. In the DEACON work the bull was taken by the horns, so to speak. It has been found that a wide variety of forms can be accommodated in a reasonable number of rules. Although computing time due to parsing is still a critical problem, even here times are achieved which make the result feasible for a number of applications.

What about ambiguity? It is well known that systems for the syntactic analysis of natural languages produce an unacceptably high number of ambiguous syntactic analyses for a given sentence. This is to some extent true in the DEACON syntactic analysis as well. However, systems built along lines described herein go beyond syntactic analysis. It is found in practice that the semantic analysis aspects of the

system resolve many of these syntactic ambiguities. In many cases, several parsings will yield a single meaning. More often, the interpretive rules, when applied to a parsing, will indicate it to be semantically vacuous, thereby reducing the number of meaningful analyses.

However, ambiguities remain. In some areas of computing, areas indeed which currently account for the great bulk of computations, a single program will be used to make a large number of calculations, and speed of processing and precision of statement are prime requirements. In this case, an algebraic language allowing no ambiguities, with an optimizing compiler to produce an efficient object program, is certainly called for. Even here the question of ambiguities at the problem definition level, before the programmer begins his translation, cannot be wholly overlooked.

When the ultimate user is less clear concerning his problem and the computer enters into the creative feedback loop, there is great advantage in providing the means for communication directly with the computer in a language he finds natural and which has greater flexibility. Further, there is a vast area where the computer can be of great value to ongoing operations, where military and management staffs need effective access to data in forms responsive to their immediate needs. The expression of these data manipulating requirements to the computer differs only by degree from programming as the computer specialist knows it. It is in these latter categories of programming that the programming language should be English. The conversational mode provides the means for immediately resolving ambiguities. The advantages of the interpretive mode for immediate response are not over balanced by the need for optimized code. And the naturalness of the language frees the user for concentration on the problem at hand rather than on its translation.

REFERENCES

1. For recent comment on the problem and prospects of "English as a Programming Language" see the discussion between Jean Sammett, R. W. Floyd et al in *Comm. ACM*, vol. 9, pp. 228–30 (1966).
2. A. Tarski, "Der Wahrheitsbegriff in den formalisierten Sprachen," *Studia Philosophica*, vol. 1, pp. 261–405 (1936); English translation in *Logic, Semantics, Metamathematics*, Oxford University Press, New York, 1956, pp. 152–278.
3. See papers of Turing and Gödel in M. Davis (ed.), *The Undecidable*, Raven, New York, 1956.
4. R. F. Simmons and K. L. McConlogue, "Maximum-Depth Indexing for Computer Retrieval of English Language Data," *Amer. Documentation*, vol. 14, pp. 68–73 (1963).
5. —, S. Klein, and K. L. McConlogue, "Indexing and Dependency Logic for Answering English Questions," *ibid*, vol. 15, pp. 196–204 (1964).
6. E. Irons, "A Syntax Directed Compiler for ALGOL 60," *Comm. ACM*, vol. 4, pp. 51–55 (1961).
7. N. Wirth and H. Weber, "EULER: A Generalization of ALGOL, and its Formal Definition," *ibid*, vol. 9, pp. 13–25, 89–99 (1966).
8. E. L. Post, "Formal Reductions of the General Decision Problem," *Am. J. of Math.*, vol. 65, pp. 197–215 (1943).
9. J. McCarthy, "A Basis for a Mathematical Theory of Computation," in P. Braffort and D. Hirschberg, *Computer Programming and Formal Systems*, North Holland, Amsterdam, 1963, pp. 33–70.
10. K. Gödel, *The Consistency of the Axiom of Choice and of the Generalized Continuum-Hypothesis*, Princeton University Press, 1940.
11. N. Chomsky, "Three Models for the Description of Language," *IRE Transactions on Information Theory*, IT-2(3), pp. 36–45 (1956).
12. —, and G. A. Miller, "Introduction to the Formal Analysis of Natural Languages," in R. D. Luce, R. Bush, and E. Galanter (eds.), *Handbook of Mathematical Psychology*, vol. II, Wiley, New York, 1963, pp. 269–322.
13. Z. S. Harris, "Transformational Theory," *Language*, vol. 41, pp. 363–401 (1965).
14. A. Tarski, "The Semantic Conception of Truth and the Foundations of Semantics," *Phil. and Phenomenological Research*, vol. 4, pp. 341–76 (1944); reprinted in H. Feigl and W. Sellars (eds.), *Readings in Philosophical Analysis*, New York, 1949.
15. D. T. Ross and J. E. Rodriguez, "Theoretical Foundations for the Computer-Aided Design System," *Proc. of Spring Joint Computer Conference*, 1963, pp. 305–22.
16. J. E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," *ibid*, pp. 329–346.
17. B. F. Green, Jr., et al, "Baseball: An Automatic Question Answerer," *Proc. of Western Joint Computer Conference*, 1961, pp. 219–24.

AN APPROACH TOWARD ANSWERING ENGLISH QUESTIONS FROM TEXT *

R. F. Simmons, J. F. Burger and R. E. Long

*System Development Corporation
Santa Monica, California*

INTRODUCTION

Research on question answering by Raphael,¹ Black,² and Elliott,³ and our own work on Proto-synthex II⁴ has shown that question-answering algorithms can be most easily written if the text source is in the form of simple, explicitly structured sets of subject-verb-nominal strings. Question-answering algorithms that have thus far been developed include word- and structure-matching operations and some few logical inference functions. All of the systems cited have in some fashion limited their input language to simple subject-verb-nominal strings, thus eliminating many problems of syntactic analysis and providing a normalized form for language data.

Our approach to question answering from natural-language text requires that both text and question be normalized into standard subject-verb-nominal kernels. Determining whether a set of text kernels answers a question or not is a complex matter of applying meaning-preserving transformations, to discover if pairs of apparently unlike kernels are in fact synonymous. The essential feature of the re-

search reported here is an approach to question answering that depends on the discovery of sets of equivalence operators that can determine whether or not two English language strings are meaning-preserving paraphrases of each other.

AUTOMATIC DERIVATION OF KERNELS

Recently Kuno⁵ and Foster⁶ have developed algorithms that can derive kernel sentences from English text that has been analyzed by the Harvard Syntactic Analyzer. Joshi⁷ and a group at the University of Pennsylvania have also developed what may prove to be programmable algorithms for kernelizing. None of these algorithms is in completely programmed form as yet, nor are the definitions of kernels that have been so far developed completely suited to question-answering.

We have constructed a system that produces a type of text-normalizing kernel that we believe is more satisfactory for the purpose of answering questions from English text. We base our derivation of kernels on Chomsky's⁸ theory of deep syntactic structures of sentences. For each phrase marker headed #S# of the underlying deep structure of a sentence, the kernel maker is required to produce a kernel string. The kernel strings are of the form $X_1 R X_2$ where X_1 and X_2 are usually in the form of nouns but may be reference numbers to other

* The research reported in this paper was sponsored by the Advanced Research Projects Agency Information Processing Techniques Office and was monitored by the Electronic Systems Division, Air Force Systems Command, under contract AF 19(628)-5166 with the System Development Corporation.

kernels, adjectives, adverbs, or null, and where R is a relational term ranging from simple verbs such as be, have, run, to a complex verb-particle or verb-preposition combination such as be-of, run-off, fly-from, etc. A null term (symbolized N') in an X position is typified by the "someone" or "it" of an incomplete passive construction. In the case of certain sentences such as "John wants to swim the Channel" the X₂ position of the kernel #1, "John wants 2," must refer to a second kernel #2, "John swims the channel." It is convenient to treat X₁ terms in a similar fashion in such sentences as "That she will come is delightful."

In addition to such referencing within kernels from one to another, each kernel is associated with a cross-referencing term that records and labels its relation to coordinate or immediately superordinate kernels of the sentence. The intent of the cross-referencing is to maintain a labeled tree structure for the set of kernels that has been extracted from the sentence. Labels include conjunctions, relative pronouns, and indications of adjectival, prepositional and other types of subordination. For example in the sentence "John or Mary went to the store that sold skis," the following kernels with reference terms are derived:

1. John went-to the store (or 2)
2. Mary went-to the store (or 1)
3. Store sold skis (that 1, 2)

The referencing scheme is designed to allow for the complete reconstitution of the original sentence and its surface structure. The reference terms will be seen below to be required for some question-answering operations.

Underlying the kernel-maker we use PLP II, which is a parsing system that produces (among other things) an immediate constituent analysis for a wide range of complex English text. Where phrase structure rules are inadequate to produce the IC analysis (particularly as in questions) a limited number of transformational rules have been included in the grammar. The form and operation of the PLP II system is described in Burger, Long and Simmons.⁹ It is important in regard to the kernel maker to note that it produces multiple analyses for most sentences that are given to it. However these are each in the form of a single structural description as illustrated in Fig. 1. The fact that only single unique descriptions are passed on to the kernel maker greatly simplifies its task. Our concern in this research is not

with resolving syntactic ambiguity but with deriving kernel strings from the surface structure description of sentences.

Our procedure for obtaining kernels from the surface SD (structural description) of a sentence is outlined below. Although there may be room for argument that kernels representing deep structures can be obtained directly from the surface SD of the sentence, we interpret the procedures used by Kuno,⁵ Foster,⁶ and Lieberman¹⁰ and our own experience with recent experiments in kernelization to indicate a strong affirmative to the notion.

1. Given a complete surface SD of a sentence as in Fig. 1(c), scan it to discover structural indices of kernels. A structural index (SI) for a kernel is a three-node structure A(B C) such that B and C are both nodes dominated by A. Thus NP(N PP), NP(NP PP), NP(Adj. N), S(NP VP) are all examples of structural indices that identify kernels. NP(Art. N), VP(V NP), VP(V PP) and PP(Prep. N) are examples that do not.

JACK AND JILL WENT UP THE HILL AND FETCHED A PAIL OF WATER .	
(2 PARSINGS)	
1	
(1 (JACK N AND 2)	
(2 AND CONJ AND 8)	
(3 JILL N AND 2)	
(4 WENT V AND 8)	
(5 UP PREP WENT 4)	
(6 THE ART HILL 7)	
(7 HILL N UP 5)	
(8 AND CONJ ' ∅)	
(9 FETCHED V AND 8)	
(10 A ART PAIL 11)	
(11 PAIL N FETCHED 9) (12 OF PREP PAIL 11) (13 WATER N OF 12)	
a) <u>Dependency Analysis</u>	
TREE	
b) <u>Dependency Tree Structure</u>	
((AND (AND (JACK) (JILL))	
(WENT (UP (HILL (THE)))) (FETCHED (PAIL (A) (OF (WATER))))))	
c) <u>Immediate Constituent Structure Description</u>	
I CL	
(5 (NP (N JACK) (HNP (CONJ AND) (N JILL)))	
(VP (VP (V WENT) (PP (PREP UP) (NP (ART THE) (N HILL))))	
(HVP (CONJ AND)	
(VP (V FETCHED)	
(NP (ART A) (NP (N PAIL) (PP (PREP OF) (N WATER))))))	
d) <u>Kernel Structures</u>	
KERNEL	
1 ((JACK WENT UP THE HILL) and #3	
2 (JACK FETCHED A PAIL) and #1	
3 (JILL WENT UP THE HILL) 4 (JILL FETCHED A PAIL) 5 (PAIL (IS OF) WATER))	

Figure 1. Examples of computer-produced syntactic analysis from PLP II, showing simple kernels.

2. Look up the SI in a list of kernel structural indices (KSI) to discover if it is a KSI. If not, continue the scan. If it is a KSI, there will be associated with it a function that operates on the SD of the sentence to produce, first a kernel SD, then a kernel string. The functions are exactly equivalent to inverse transformational rules.

For example given the KSI, NP(Adj. N), the associated function transforms it to S_k (N be Adj.) which is the SD of a kernel string. (S_k stands for kernel sentence.) A more complicated case is illustrated by the KSI, NP(rel. pron. VP). Here the function is required to find the noun antecedent of the relative pronoun and substitute it into the resulting kernel string. This occurs in two steps, which may be most easily shown in a transformational notation:

- #1 $NP((\dots N_1 \dots) (NP(\text{rel. pron.} + VP))) \rightarrow NP((\dots N_1) (NP(N_1 + VP)))$
 #2 $NP(N_1 + VP) \rightarrow K_s (N + VP)$

The K_s is now able to generate a kernel string.

3. Label each kernel. The labels for each kernel correspond to specific features of the KSI functions that generate them. The label for the last illustration would include the actual relative pronoun. The label for a prepositional or adjectival kernel would be "prep." and "adj.," respectively.
4. Reference each kernel to the kernel or kernels that are coordinate and superordinate to it. This information is available as a result of the order of processing from deepest structures upward and from the labels previously assigned.

So far we have experimented with LISP functions that accomplish steps (1) and (2) and, though we can see that (3) and (4) are not difficult in principle, we have not yet fixed on a particular algorithm. It will be noticed that we do not follow transformational programming conventions of the type adopted by the MITRE system or by Foster. The reason is that in LISP notation, the SD of a sentence is rela-

tively easy to deal with directly and with generality. Chomsky,⁸ the MITRE group, Foster⁶ and our own experience indicate that relatively few KSIs and functions will be required to generate kernels from a wide range of English sentences.

Serious problems are expected in recovering kernels from those surface structures that have resulted from the repeated application of deletion transformations. Additional problems are expected in developing fully adequate notation for cross-referencing and labeling the kernels from a sentence. Problems involving the need to find antecedents for pronouns and other pronominal expressions will probably also have to be dealt with. In this latter regard rules for resolving anaphora developed by Olney and Londe¹¹ have proved helpful.

Other problems concern our definition of the R (relational) term of the kernel—we are by no means certain that our method of dealing with verbs, adverbs, and prepositions is the best. We expect further light to be shed on this problem as a result of attempting to use the kernels for answering questions.

ANALYSIS OF QUESTIONS

Our basic approach to question answering is to make a surface structure analysis of the question, resolve it into kernels,* and then compare the kernels resulting from the question with those stored in a directed graph structure. The stored data are obtained by kernelization of English sentences that have been given to the system. As in the approach to kernelizing above, PLP II is used to obtain a surface parsing of the question.

The comparison of Question kernels (Q-kernels) with Text kernels (T-kernels) will be a simple comparison only in the case that the question is *complete*,¹³ that is, of the form "Aux V NP VP," or "is NP NP," (e.g., do NP VP, can NP VP, is NP NP), and there exists an exactly corresponding T-kernel. Generally, the problem will be one of discovering if an appropriate set of T-kernels can be found in the data structure and transformed into a match with the set of Q-kernels derived from the question.

In perhaps the simplest case, the question "Is X a Y?" is first resolved into the Q-kernel, X be Y.† X and Y are used to search the headings of the data

* This approach was discussed earlier by Walker and Bartlett.¹²

† This discussion assumes "N be N" kernels to select the class membership sense of be. Kernels of the type "N be Adj." follow a different line of logic.

structure (described in Simmons ⁴). If the T-kernel X be Y is discovered there is no problem. If the T-kernels X be W , W be V , B be Y are discovered, the fact that "be" is known to be logically transitive allows X be Y to be derived, again answering the question. This logic applies also for such obviously transitive relations as farther than, nearer than, brother of, part of, etc. (see Raphael,¹ Cooper,¹⁴ and Black ²).

If no transitive relation is found to lead from X be W to X be Y , the synonym operation of inferring that $W \cong Y$ may show that $(X \text{ be } W) \rightarrow (X \text{ be } Y)$, and again an answer is possible. The operations illustrated in this simple question are almost the only ones that have been used in question-answering systems up to the present.* They are obviously not sufficient to answer more than a tiny subset of questions. The more ambitious systems ^{16,17} have devoted a great deal of effort to dealing with analysis of the question into its request structure, its syntactic structure, finding word and structure matches, using synonyms, etc., but they have dealt only superficially with what we see as the central problem of transforming from a set of relevant T-kernels \ddagger into the exact form of the Q-kernels.

More generally the problem can be described as follows:

Given the set of
 Question kernels, $Q_1 = \{KQ_1 KQ_2 \dots KQ_n\}$
 and the set of Text kernels, $T_1 = \{KT_1 KT_2 \dots KT_m\}$
 If there is a set of
 transforming operators, $TO = \{TO_1 TO_2 \dots TO_k\}$
 such that, $Q_1 = TO \times T_1$,
 or, $T_1 = TO \times Q_1$,

than Q_1 has a complete answer.†

Discovering operators of the set TO and the conditions under which they may be applied to English words appears to be the basic research problem in question answering.

In Belnap's ¹³ logic of questions, a question may be analyzed into two parts, a *request* and a *set of*

* However, as exceptions, Bobrow ¹⁵ used implication in his algebra problem-solver and Elliott ³ studied the use of reflexivity, symmetry and other properties of relational words in a question answerer.

† A T-kernel is defined as *relevant* if one or both of the N's in its N-R-N structure is the same as, or can be transformed into, an N from a Q-kernel.

‡ Also a measure can be applied to Q_1 and to $TO \times T_1$ to discover the degree to which $TO \times T_1$ corresponds to Q_1 . This is a measure of the completeness of the answer. The percent of word and structure matches measured in Protosynthes I, is a crude first approximation to such a measure.

alternatives. The request can usually be recognized in such forms as "what," "what (noun)," "what are (number)," "where," "when," "how," "how many," etc. The set of alternatives may be the remainder after the request kernels are removed.

Given a question: "What are the paths of rockets or missiles called?" the following set of Q-kernels can be derived:

- #1) Paths are what?
- #2) Someone calls paths.
- #3) Paths are-of rockets.
- #4) Paths are-of missiles.

There is obviously a request kernel, #1), and two kernels specifying the set of alternatives. (Whether or not it will always be as easy to separate request kernels and alternatives is not known but it is doubtful.) The alternatives are used in a question-answering system as index terms to find relevant information. The request is used to evaluate the information (e.g., "how many" requires a number, "where" requires a place coding, "when" a time coding, etc.) and sometimes also to process it (how many, or the commands, "list 7, name 3, etc." require counting or listing functions).*

Although both request kernels and alternative kernels must be identified so that we can understand how to use them, the main difference seems to be that the request kernel definitely indicates a set of operations to be performed on the answer kernels while the alternative kernels imply a set of operations to be performed on the data base. It may be that these operations may be conceived of in the same framework as the transforming operations required to match sets of T-kernels to a set of Q-kernels (with a consequent simplification of the whole system).

LINGUISTIC INFERENCE VIA EQUIVALENCE-OPERATORS

We see the basic problem of question answering to be one of discovering equivalence-operators and describing the conditions under which they can be applied to make Q- and T-kernels identical. Chomsky (p. 162),⁸ among others, offers the following two sets of sentence-pairs as examples of paraphrases that are not accounted for by identical deep syntactic structures:

* In this research we are concerned only with fact retrieval questions and are ignoring the algebra problem or questions typified by "How many letters are in *Oliver Goldsmith*?"

1. John strikes me as pompous.
I regard John as pompous.
2. I liked the play.
The play pleased me.

The second pair is in the form of kernels and it can be seen that some form of inverse operator * can be applied to account for the paraphrase:

$$(N_1 \text{ likes } N_2) \times TO_1 = (N_2 \text{ pleases } N_1)$$

where: "×" signifies the application of the operator, and "=" is interpreted to mean "functionally equivalent in context." The operation appears to apply for kernels involving "like" as a verb in the syntactic context of nouns.

In set 1, however, the situation appears to be more complex. In the first place it applies to a complex three-part relationship as follows:†

$$(N_1 \text{ strikes } N_2 \text{ as Adj.}) \times TO_{11} = (N_2 \text{ regards } N_1 \text{ as Adj.})$$

This equivalence operation appears to apply without exception with reference to strikes-as: regards-as but it does not apply to the kernel (N_1 strikes N_2) especially where the striking is accomplished with a club.

Apparently the inverse-equivalence operator is a very common occurrence in English in such pairs as bought-sold, gave-received, read-write, etc., as well as in the active-passive syntactic transformation. There appear to be many cases of its application where only syntactic conditions restrict its use.

Another frequent case of paraphrase that has proved important in question answering is the matter of substituting synonyms. If we consider a synonym-equivalence operator, T_s , there are some cases such as:

$$(N_1 \text{ eats } N_2) \times T_s = (N_1 \text{ devours } N_2)$$

where the equivalence may be sufficient for practical purposes but where there are probably always semantic restrictions limiting the substitution. Sparck Jones¹⁸ discusses this point in detail.

There is also a large set of weak implication operators that may be applied to English statements but apparently only under highly specified semantic and pragmatic conditions. For examples:

* Inverse $(R_1R_2) = V_{x,y} (x R_1y \rightarrow y R_2x)$

† It is of interest to note how easily this applies to a five-part nonkernel in contrast to the cumbersomeness of applying it to two kernels.

- ? A eats B ± B is inside of A
- ? A flew-from B ± B was at A
- ? A is-made-from B ± B is-part-of A
- ? A is-above B ± B is-below A
- ? A struck B ± B received a blow.

The question marks and the symbol "±" indicate our ignorance of the conditions under which these equivalences may be valid.

A more satisfying set of operations has to do with class membership. If A is a member of the class B, $A \rightarrow B$.

Thus:

walk	→	move
go	→	move
aardvark	→	mammal
mammal	→	animal
steak	→	meat
meat	→	food

However such relationships are not easy to discover for many if not most English words.

From our work in question answering we are dimly aware that there are many complicated equivalence operations involved in the various ways that answers to a given question may be paraphrased. The matter of matching Q-kernels to T-kernels will prove a very complex process requiring not only the discovery of appropriate equivalence operators and the syntactic and semantic conditions of their use, but also an understanding of the logical properties (e.g., transitivity, symmetry, asymmetry, reflexivity, etc.) of relation terms to further define their application. In our work we concentrate mainly on the discovery of operations that depend only on syntactic restrictions and believe that these may prove to be sufficient to greatly increase our understanding of the question-answering process.

PROPOSED DISCOVERY PROCEDURE FOR EQUIVALENCE-OPERATORS

Our approach to the study is to select a broad range of question types with varying forms of answering statements associated to each question. Questions and answers will be reduced to kernels (by hand if necessary or by the kernelizer described above). We will attempt to discover a set of equivalence-operators that can transform the T-kernels into Q-kernels. Such operators will be tested on larger samples of English text by being translated into LISP functions and tested in the context of Protosynthex II.

It is our hope that the set of equivalence operators that are discovered and the LISP functions that embed them may eventually form the basis of a powerful question-answering and paraphrasing language.

The questions will be selected from the *Modern Science Quiz Book*,¹⁹ which offers a wide range of syntactic forms and semantic content. Although we do not propose any rigidly specified sampling procedure, we intend to pay particular attention to the more difficult and complicated questions represented in this compendium.

In selecting answer sets for each question, we will depend on encyclopedias such as *Compton's* and the *Golden Book* to insure a wide and varied range of paraphrases. We expect also to compose answers in some cases, and to use answers made up of more than one sentence.

Table 1 shows a question Q1, the set of its Q-kernels and two alternate answers A1 and A2 and the sets of T-kernels derived from them. We will describe some operations that can be applied to the Q-kernels and to the T-kernels that can serve to make them isomorphic. Since this theory of question answering has not yet been empirically tested, we do not pretend that this discussion or the resulting operators are more than illustrative—they are included only to exemplify the line of approach.

The kernel Q1.1 may first be subject to a deletion operator. The basis for this is that "N' calls N" is a kernel (where N' is deleted after the passive transformation) which shows that "calls" is used in a meta-structure that is not necessarily relevant to answering the question.* After this operation the kernels A2.1 and A2.3 form a sufficient answer (assuming "is = are" and recognition of the meaning of the "or" relation between Q1.2 and Q1.3). The resulting answer is in the form "Paths are trajectories, paths are of missiles," or some combination of these into a more complex sentence.

To discover how A1 is also an answer is more difficult. As native speakers of English we understand that in kernel A1.1 it is legitimate to transform "rockets leave paths" into "paths are-of rockets"; on the other hand if the kernel were "rockets leave planets" it is not at all satisfactory to say "planets are-of rockets." Thus, if it is possible to transform "N1 leave N2" into "N2 are-of N1," there must be

* This assertion must be worked out in more detail for "calls," "names," etc.

Table 1. Q-Kernels and T-Kernels from a Question and Two Possible Answers

Q1	What are the paths of rockets or missiles called?
Q1.1	N' calls paths
Q1.2	Paths are what
Q1.3	Paths are-of rockets
Q1.4	Paths are-of missiles
A1	Rockets leave fiery paths as their trajectories.
A1.1	Rockets leave paths
A1.2	Paths are-of fire
A1.3	Paths are-as trajectories
A1.4	Trajectories are-of rockets (assumes pronoun substitution)
A2	The path of a bullet or missile is its trajectory.
A2.1	The path is trajectory
A2.2	Path is-of a bullet
A2.3	Path is-of a missile
A2.4	Trajectory is-of a bullet (assume pronoun substitution)
A2.5	Trajectory is-of a missile (assume pronoun substitution)

semantic conditions limiting the N1-N2 pairs to which equivalence applies.

Ignoring this possibility, kernel A1.3 appears more tractable. "N1 are-as N2" appears to lend itself invariably by deletion operation, T_{as} , into "N1 are N2." (Reasons for this have been found in considering "as" in certain environments as a marker of discourse equivalence by Olney (unpublished).

By using:

$$T_{as} \times (N1 \text{ are-as } N2) \rightarrow (N1 \text{ are } N2),$$

kernel A1.3 now states "Paths are trajectories." Since "is" has the property of transitivity, it follows that we can equate "trajectories" and "paths" in A1.4 to get a new kernel A1.5 "paths are-of rockets." Now a match is available as follows:

- Q1.1 (N' calls paths) = ϕ
 Q1.2 (Paths are what) = $A1.3T_{as}$ =
 (paths are trajectories)
 Q1.3 (Paths are-of rockets) = $A'1.5$ =
 (paths are-of rockets).

The discussion essentially illustrates our approach to a discovery procedure for finding equivalence operators. The importance of translating these into programs and testing them on larger samples of text can hardly be overrated. In the first place, the need to program them insures a complete understanding of what is the operational meaning of such easy phrases as "equating 'trajectory' and 'path' on the

basis of the transitivity of 'is'."* In the second place the program makes them more easily testable against larger sets of text so that a fair assurance can be developed that a final set of equivalence operators is of wide generality.

In summary, we propose to test the theory by developing sets of equivalence rules that can be used to transform T-kernels from statements that are answers to questions into the form of the Q-kernels from the questions. As these rules are developed, they will be expressed in the form of LISP functions and used in the context of the Protosynthes II question-answering system to test their validity on a wider range of questions.

ACKNOWLEDGMENTS

We are grateful to Dave Londe and John Olney for their critical reading and helpful suggestions and to Bruce Fraser, whose critical comments on a first draft of this paper were extremely helpful.

REFERENCES

1. B. Raphael, "SIR: A Computer Program for Semantic Information Retrieval," *Proc. Fall Joint Comput. Conf.*, vol. 25, Spartan Books, Washington, D.C., 1964. (Also available as a doctoral dissertation, Math. Dept., MIT, 1964.)
2. F. S. Black, "A Deductive Question-Answering System," doctoral dissertation, Div. Eng. and App. Phys., Harvard Univ., 1964.
3. R. W. Elliott, "A Model for a Fact Retrieval System," doctoral dissertation, Univ. of Texas Comput. Ctr., 1965.
4. R. F. Simmons, "Storage and Retrieval of Aspects of Meaning in Directed Graph Structures," *Commun. of the ACM*, vol. 9, no. 3, p. 211-15 (1966).
5. S. Kuno, "A System for Transformational Analysis," Rep. NSF 15, Comput. Lab., Harvard Univ., (1965).
6. D. R. Foster, "Automatic Sentence Kernelization," *ibid.*
7. A. Joshi, "A Transformational Decomposition Program," paper presented at Inform. System Colloq., NSF, 1966.
8. N. Chomsky, *Aspects of the Theory of Syntax*, MIT Press, Cambridge, Mass., 1965.
9. J. F. Burger, R. E. Long, and R. F. Simmons, "An Interactive System for Computing Dependencies, Phrase Structures and Kernels," SDC document SP-2454, System Development Corp., Santa Monica, Calif. (1966).
10. D. Lieberman et al, "Automatic Deep Structure Analysis Using an Approximate Formalism," *Studies in Automatic Language Processing*, AFCRL-65-680, 1965, pp. 1-73.
11. J. C. Olney and D. L. Londe, unpublished communication, 1966.
12. D. E. Walker and J. M. Bartlett, "The Structure of Languages for Man and Computer: Problems in Formalization," *First Cong. on Inform. Sci.*, 1962.
13. N. D. Belnap, Jr., "An Analysis of Questions: Preliminary Report," SDC document TM-1287, System Development Corp., Santa Monica, Calif. (1963).
14. W. S. Cooper, "Fact Retrieval and Deductive Question-Answering Information Retrieval Systems," *J. of the ACM*, vol. 11, no. 2, pp. 117-37 (1964).
15. Bobrow, D. G., "Natural Language Input for a Computer Problem-Solving System," *Proc. Fall Joint Comput. Conf.*, vol. 25, Spartan Books, Washington, D.C., 1964. (Also available as doctoral dissertation, Math. Dept., MIT, 1964.)
16. B. F. Green, Jr., et al, "Baseball: An Automatic Question Answerer," *Computers and Thought*, McGraw-Hill, New York, 1963, pp. 207-16.
17. R. F. Simmons, "Synthetic Language Behavior," *Data Processing for Mgmt.*, vol. 5, no. 12, pp. 11-18 (1963). (Also available as SDC document SP-1245, System Development Corp., Santa Monica, Calif., 1963.)
18. K. Sparck Jones, *Synonymy and Semantic Classification*, Cambridge Language Research Unit M.L. 170, Cambridge, England, 1964.
19. O. E. Epple and L. E. Epple, Jr., *Modern Science Quiz Book*, Platt and Munk, New York, 1958.

* e.g., Transitivity of $R = \forall x,y,z (x R y \wedge y R z \rightarrow x R z)$.

DEACON: DIRECT ENGLISH ACCESS AND CONTROL *

James A. Craig
Susan C. Berezner
Homer C. Carney
Christopher R. Longyear

General Electric Company, Santa Barbara, California

INTRODUCTION

The extensive syntactic ambiguity inherent in natural language has been convincingly shown by such systems as the Harvard syntactic analyzer.¹ Furthermore, no semantic techniques are in prospect for satisfactory resolution of this ambiguity by computer. In contrast, well-developed semantic techniques exist for formal languages.

In an accompanying paper,² Thompson defines a formal language and a technique for determining the meaning of sentences in that language. The semantic technique is to use *interpretation rules* which define actions (or sequences of actions) involving the objects of an *environment*. An environment is defined as a finite set of categories of computer memory structures. Thompson hypothesizes that English essentially becomes a formal language as defined if its subject matter is limited to "material whose interrelationships are specifiable in a limited number of precisely structured categories [memory structures]."

The DEACON system constitutes a test (and, we

feel, a confirmation) of that hypothesis.† The environment, in this application, consists of "ring" structures in which data is stored as it is introduced into the system. The interpretation rules are computer programs that perform various operations on the ring structures.

Because these programs are written in terms of structural categories (independent of content), the interpretation rules apply to any subject matter that is stored in these categories.‡ Each interpretation rule is associated with a rule of a context-sensitive phrase structure grammar. A combination phrase structure rule/interpretation rule determines the meaning of the phrase that is formed by applying the rule. A sequence of rules applied by a parsing routine determines the meaning of an English sentence, phrase by phrase, in accordance with the subject matter of the data base. Simmons has called this "parsing directly into a data structure."⁵

† Not only did Thompson develop the theoretical basis for DEACON, but he also directed and participated in all aspects of its application before joining the staff of the California Institute of Technology.

‡ This is the major advance of DEACON over Green's BASEBALL³ and Lindsay's SAD SAM.⁴ These earlier systems provided valuable background for DEACON, and discussions with Lindsay on DEACON itself were extremely helpful.

* The DEACON project is primarily supported by the Rome Air Development Center under contract AF30(602) 4272 and also by the Research and Development Center of the General Electric Company.

The question of subject-matter limitations (which are necessary in order to use formal-language semantic techniques on English) is crucial to the effectiveness of a DEACON system. The severity of the limitation depends inversely on the adequacy of the memory structures that are used. Among the more advanced structures yet devised are ring structures, such as those used by Sutherland in SKETCH-PAD.⁶ The particular ring structures now used in DEACON (devised by Thompson, with contributions by R. Donald Freeman, Jr.) are described later in this paper.

Ring structures are adequate for storing a wide range of richly interrelated data that is pertinent to such functions as intelligence analysis, management planning, and decision making. Typical of these functions are resource allocation problems, in which the pertinent data is an inventory of the resources, their characteristics, and their interrelations. This type of data is specifiable in ring structures.

It is for such management functions that DEACON is being developed. DEACON as a management information system has the following characteristics: Any particular management staff works with a private data base relevant to its own operations, rather than with a universal or pre-established data bank. A user puts data into the system by typing appropriate English statements at a teletype in his normal working area. Similarly, he elicits data by typing English queries. Whether inputting data or asking questions, the user need not know how the data is stored. The interpretation rules automatically relate English statements to the data structures stored in computer memory. This permits the user to concentrate on his problem rather than irrelevant detail.

The use appropriate for a DEACON system differs in a number of important respects from uses which might be thought appropriate for large, fixed-format systems. Characterization of human informational processes suggests the theoretical inadequacy of fixed-format systems in any real application and also predicts that the most efficient informational processes occur only within a small informational community concerned with shared subject matter in a rapidly changing context. The intended use of a DEACON-type system based on augmenting human informational processes is discussed more fully in references 7 to 11. These references present both the philosophy of informational processes and illustrations of the anticipated use of DEACON-type systems.

This paper is intended to illustrate the application of the theory presented in Reference 2. It describes the use of interpretation rules by the DEACON system in analyzing and responding to English sentences on the basis of stored data. The following section shows the functioning of a typical rule in abstraction and then illustrates its use along with others in analyzing a sample sentence. Ring structures and how they are built (data input rules) are then described, followed by a description of the use of a "verb table" in analyzing certain types of sentences. Next is a description of the parser that applies the grammar rules, followed by a brief description of the hardware configuration used and then some conclusions. An appendix describes some more grammar rules and lists sample sentences and corresponding system responses.

INTERPRETATION RULES IN SENTENCE ANALYSIS

To analyze a sentence, the DEACON system must be able to recognize the strings of characters that form the sentence. To accomplish this, a dictionary of vocabulary terms is built up from definitions typed in by the user (currently in an interactive mode of operation but using formatted statements). A vocabulary term may be a word (a string of characters between blank characters in a sentence) or an idiom (a string of two or more words, such as San Francisco).

Because of the nature of the subject matter of a DEACON system, most of the vocabulary terms are those normally considered to denote objects, their characteristics and their interrelations. In DEACON, however, these terms denote structures in the data base. The examples used in this paper assume as subject matter a simulated army environment. In the sample data base, or subject matter characterization, the term BATTALION denotes a ring containing data about battalions, and the term 638TH denotes a ring containing data about a particular battalion named the 638TH. The ring denoted by 638TH may also be denoted by a more descriptive name, such as BLUE EAGLES. Or a given vocabulary term may denote more than one ring.

The rings mentioned above are called the data referents, or referent rings, of the vocabulary terms that refer to them. The referent rings may be thought of as doors to the data, with specific data

entries consisting of connective rings intersecting appropriate referent rings, as shown later.

The terms that denote rings are called referent terms, and usually are assigned part of speech R, for ring, or V, for verb.* Function words, such as prepositions and articles, normally do not denote rings. In some areas of the system, these words are treated as a class, and are identified with part of speech F. In rules of grammar, however, each function word is its own part of speech.

Other parts of speech are: N for number; X for pronoun; T for time; S for sentence. These will be discussed as they arise.

The definition of a vocabulary term consists of a part of speech, grammatical information, and (for denotational terms) a link to a referent ring in the data base.

When a sentence is typed into the system, a dictionary lookup routine finds all the pre-defined vocabulary terms from the sentence. From each vo-

* These parts of speech indicate something about the types of structures used for storing referent data of vocabulary terms that have these parts of speech. These parts of speech therefore function as both syntactic word classes and semantic categories.⁸ In this paper, they are called either "part of speech" or "category."

cabulary term found, the system forms an *initial phrase*. In this paper, therefore, "phrase" applies to single words as well as to syntactic constructions. The DEACON representation of a phrase also includes its part of speech, grammatical information, and a link to a referent ring.

A parsing routine applies rules from the system's grammar to combinations of initial phrases to form syntactic constructions, or *intermediate phrases*. Each rule consists of two parts—the syntactic, or phrase structure, part and the semantic, or interpretation, part.

The phrase structure part of a rule specifies the part of speech symbols (categories), grammatical characteristics, and positional relations that its input phrases must have. Similarly, it specifies the part of speech and grammatical characteristics of its output phrase. For example, phrase structure Rule 5 states that two adjacent R-phrases (which may be initial or intermediate phrases, each with part of speech R) may be combined to form an intermediate R-phrase. The syntactic construction of this intermediate R-phrase is shown in tree form in Part A of Fig. 1. The fact that this part of the rule deals with part of speech *symbols* (not with the associated vo-

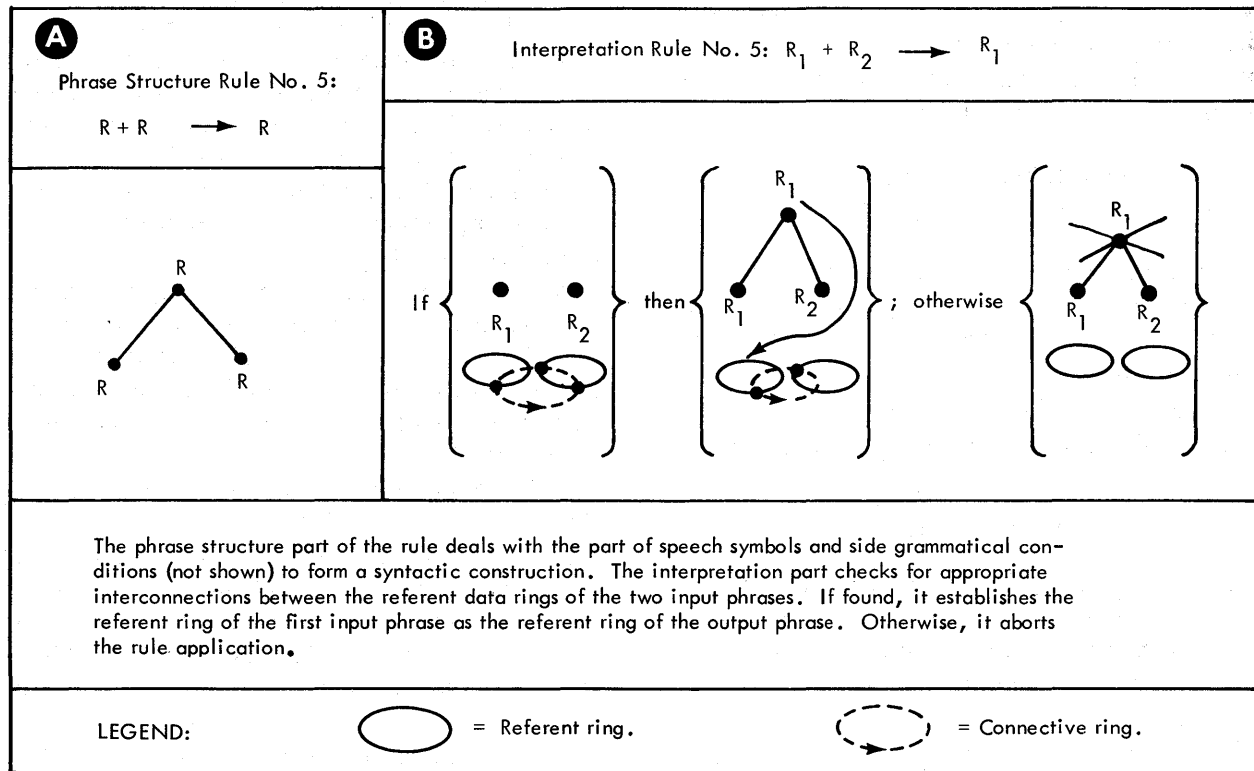


Figure 1. The functioning of a typical DEACON grammar rule.

cabulary term) is emphasized by writing the phrase structure rule as $R + R \rightarrow R$, where “+” indicates concatenation. The specific grammatical checks (for number, case, etc.) are not shown.

After phrase structure Rule 5 is successfully applied, interpretation Rule 5 is applied. The interpretation rule first determines whether the phrase being formed is meaningful in the data base by looking for a certain type (or types) of connective ring that may intersect the referent rings of the input R-phrases. In our example, a two-link connective ring satisfies the rule’s conditions, as shown in Part B of Fig. 1. Therefore, the rule establishes a link in the output phrase to a referent ring which, in this case,* is the referent ring of the first input R-phrase (shown as R_1). To show the correspondence of the referent ring of the output phrase to the referent rings of the input phrases, the rule may be written as $R_1 + R_2 \rightarrow R_1$.

If no appropriate connective ring is found, the rule application is aborted as though the phrase were ungrammatical. (Actually, the phrase is marked “VACUOUS DESCRIPTION” and is later typed out as a response if no alternative analyses result in a different response.) *Eliminating such constructions that are “grammatical but meaningless for the subject matter of the data base” is a primary technique for controlling ambiguity in the system.*

The following sentence illustrates a specific application of Rule 5 and other rules:

WHO IS COMMANDER OF THE
638TH BATTALION?

The relevant dictionary entries, data base entries, and cross references are shown in Fig. 2. The dictionary lookup produces the following internal representation of the sentence:

F R F F R
WHO IS COMMANDER OF THE 638TH

 R F
 BATTALION ?

Each segment of this representation is an initial phrase shown without its grammatical information and with the vocabulary term itself used as the link to a referent ring where appropriate.

* Other rules that have two R-phrases as constituents but which produce different output phrases are discussed in the Appendix.

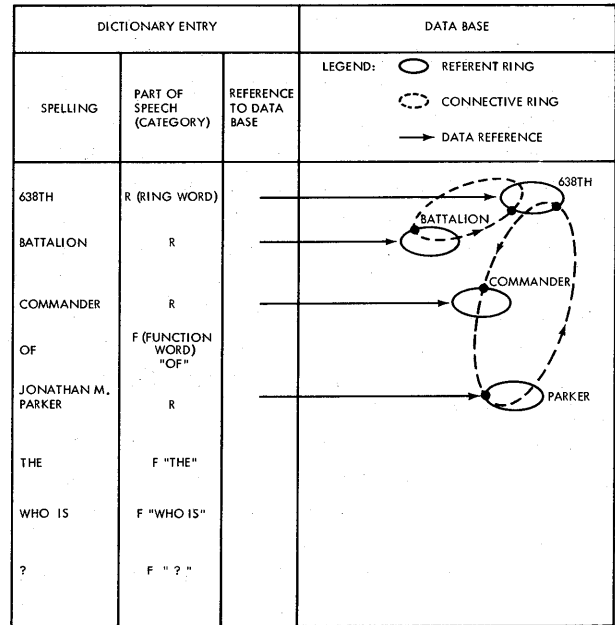


Figure 2. Sample dictionary and data base entries, and their cross references.

The parser, matching grammar rules with combinations of initial-phrase parts of speech in the internal representation of the sentence, finds that Rule 5 (Fig. 1) applies to the string “638TH BATTALION.” Its application is illustrated in Fig. 3A. Since the application is successful, the internal representation of the sentence is rewritten with the output phrase of the rule replacing the two input phrases (see Fig. 3B). In this case, the effect is the same as if the phrase BATTALION had been dropped from the original representation (after verifying that the 638TH is a battalion).

Next, the parser matches its rules against the new sentence representation. It finds Rule 11, THE + R → R (see Fig. 4). This rule in effect drops the article THE. It also illustrates the use of a function word as its own part of speech. Again, the sentence representation is rewritten.

Figure 5 shows the application of Rule 8, $R_1 + OF + R_2 \rightarrow R_3$, to COMMANDER OF 638TH. This rule introduces a new feature, i.e., that the output phrase refers to a referent ring not mentioned in the input sentence. Through the rules applied thus far, the string COMMANDER OF THE 638TH BATTALION has been replaced by its equivalent, JONATHAN M. PARKER.

The application of a final rule, WHO IS + R + ? → S, forms a sentential phrase and sets up the

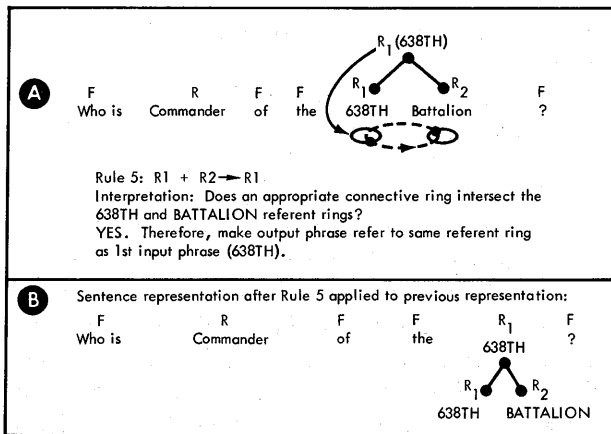


Figure 3. Examples of application of Rule 5, $R_1 + R_2 \rightarrow R_1$, and rewriting of sentence representation following it.

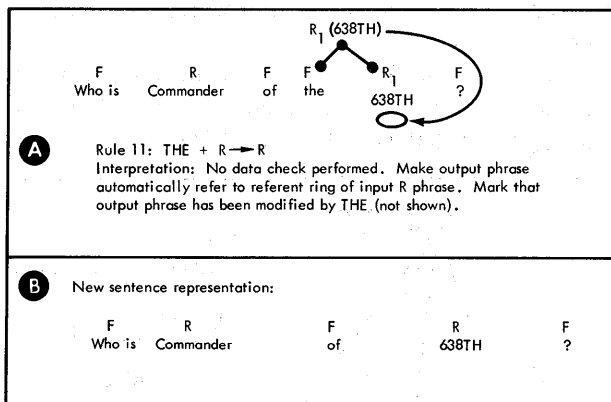


Figure 4. Example of application of Rule 11, $THE + R \rightarrow R$.

answer to the question (see Fig. 6). The system is ready to accept another message after the output routine types the answer:

WHO IS COMMANDER OF THE 638TH
 BATTALION?

JONATHAN M. PARKER

The overall sentence analysis is shown in Fig. 7. Conceptually, the system checked the data base to ensure that the unit called 638TH is a battalion, found the portion of the 638TH data record that designated its commander, and typed the associated name as the response to the sentence.

This illustration shows only one analysis of the sample sentence. However, if either syntactic or referent ring ambiguity causes alternative interpretations of the sentence, each corresponding answer is typed out. Techniques for dealing with ambiguity

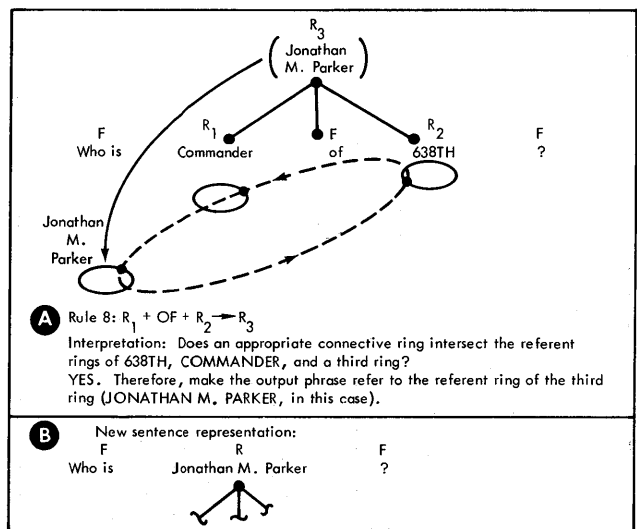


Figure 5. Example of the application of Rule 8, $R_1 + OF + R_2 \rightarrow R_3$.

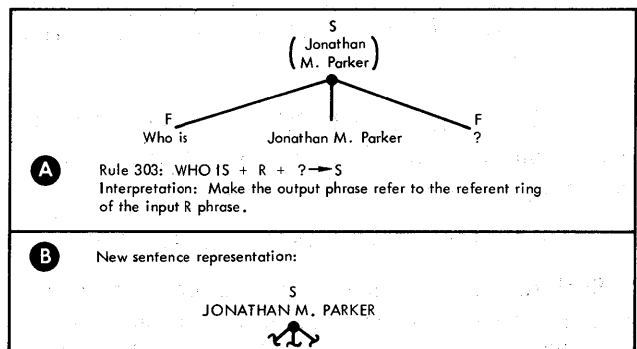


Figure 6. Example of the application of Rule 303, $WHO IS + R + ? \rightarrow S$. Since the output phrase of this rule is a single phrase with part of speech S (for Sentence), and all the elements of the input sentence are subsumed under it, its associated spelling is typed out as the answer to the input query.

are discussed in the parser section and in the Appendix.

Sentences that include a verb are handled somewhat differently. ("IS" is not defined as a verb in the above example.) Since verb processing involves more complex data structures, a sample verb analysis is not shown until after the following description of ring structures and data input rules.

RING STRUCTURES AND DATA INPUT

In the preceding example, the fact that the organizational entity called the 638TH is a BATTALION was shown by a connective ring intersecting

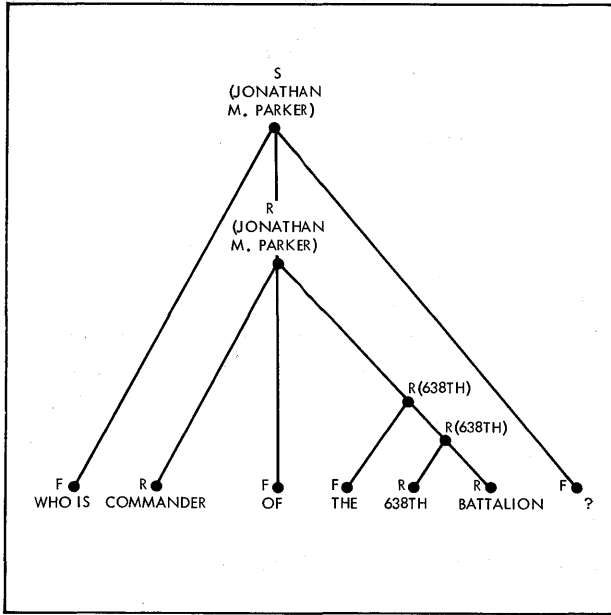


Figure 7. Tree diagram of sample sentence analysis, with nodes labeled with references to pertinent data structures.

the referent rings of 638TH and BATTALION. This connective ring (Fig. 2) was established as a data entry in response to the sentence,

DATA: THE 638TH IS A BATTALION!

Such data input sentences are parsed in the same way that query sentences are, using the same grammar. The difference comes only at the final step, where queries result in an output message, while data input statements build connective rings in the data base.* All such rules for data input begin with the word "DATA:" and end with an exclamation mark, as in Rule 901, DATA: + R + / + R + ! → S.

The slash (/) is an arbitrary function word symbol originally used in a group of formatted data input statements, such as:

DATA: 638TH / BATTALION !

These formatted statements were used to build an initial "debug data base" and are still part of the grammar. The format was relaxed slightly by adding the rule IS + A → /. Applying this rule and Rule 11, THE + R → R, to the statement shown in Fig. 8 sets up the required inputs for Rule 901,

* Rings of the data base are stored as pages on the disc and accessed from grammar rules through a paging technique.¹²

which is then executed to form the two-link connective ring.

Three-link rings are formed similarly. Samples of various ways of inputting the fact that PARKER is COMMANDER of the 638TH are shown in Fig. 9. For any of the statements to be successful, it must be parsed down to an unambiguous characterization in a form such as "DATA: COMMANDER OF 638TH IS PARKER!" If any ambiguity remains, the data entry is rejected. As illustrated, the connective ring is directed but not ordered; i.e., no "starting point" is marked. However, when such a statement is accepted, the R-word that is used attributively (COMMANDER, in this case) is placed on a special data ring associated with the preposition OF. The use of this special ring, which in effect orders connective rings such as that in Fig. 9, is discussed in the Appendix.

A variation of the three-link connective ring is introduced to store time-dependent data. For example, if the 638TH BATTALION moved from

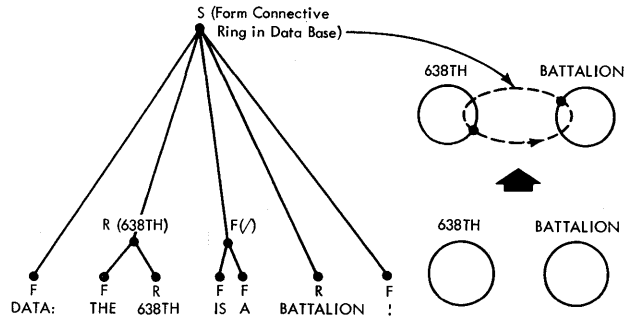


Figure 8. Example of the parsing of a data input sentence and the resulting formation of a two-link connective ring in the data base.

- DATA: 638TH / COMMANDER / PARKER !
- DATA: THE COMMANDER OF THE 638TH BATTALION IS PARKER !
- DATA: LT. COL. PARKER IS THE 638TH'S COMMANDER !
- DATA: LT. COL. JONATHAN M. PARKER IS COMMANDER OF THE ENGINEER BATTALION AT FT. LEWIS !

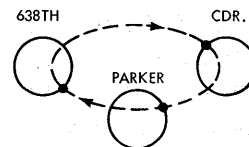


Figure 9. Examples of data input statements that result in the formation of the three-link connective ring shown.

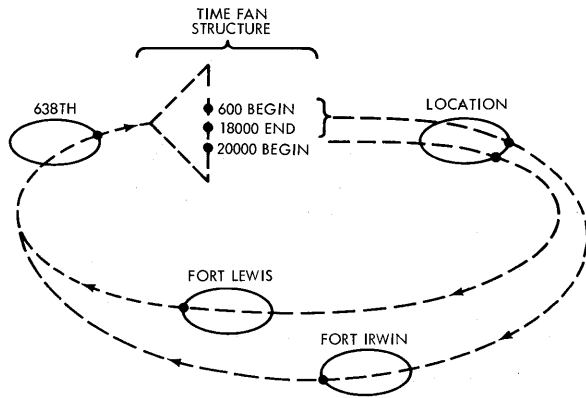


Figure 10. Use of time fan to record that the 638th Battalion was at Fort Irwin between times 600 and 18000 (points on a "time line" corresponding to particular dates) and has been at Fort Lewis since 20000.

FORT IRWIN to FORT LEWIS, this could be recorded by simply changing the connective ring. However, if a locational history is desired, a different approach is required. The technique used for recording time-dependent data involves a "time fan," as illustrated in Fig. 10.

The connective ring fans out through a "time line"* and may then lead to several different "values" for the "attribute" LOCATION. In the current system, an attribute may have only one value at a given point in time, except for the instant of change.

A time fan entry includes not only a time point, but also an "aspect" marker to show whether the applicability of the associated value is just beginning, is continuing, or is ending. For example, in Fig. 10 the 638TH has the value FORT LEWIS for the attribute LOCATION for a time span beginning at the time 20000.

Time dependent data may be input by formatted statements such as

DATA: 638TH / LOCATION / FORT LEWIS /
BEG 20000 !

Alternatively, it may be input by a sentence such as

DATA: THE 638TH BATTALION ARRIVED AT
FORT LEWIS AT 20000 !

* Since actual dates and times are of little concern in our experimental use of the DEACON system, we have to date dealt only in terms of relative numbers on a continuous time line. Lower numbers correspond to earlier dates. However, various time words such as tomorrow and next week are mapped onto the time line in relation to the recorded time of "now" and/or to the "beginning of time." "Now" is set to 0 when the system is initialized, but may be reset by typing a statement such as "IT IS NOW 20500!"

The parsing of a sentence such as this builds a "verb table," as discussed in the following section. Currently, a separate input statement is required for each entry on the time fan, and a specific time (not a span) must be given.

One other type of data structure is used. Numeric values, stored as numbers rather than as rings, result in data "dead-ends." For data such as PARKER'S SERIAL NUMBER, the number is appropriately unitless. However, the data structures used do not yet make allowance for units where they are required. Therefore, distances, weights, etc., are now stored and used as unitless numbers.

Attempts at inputting data are rejected by the system under various circumstances. Of course, any input statement is rejected if it is beyond the grammatical capability of the system. Also rejected are statements involving ambiguous data referents. For example, if there are two PARKERS in the data base, the following statement is rejected:

DATA: PARKER'S SERIAL NUMBER IS
96738332!

PARKER IS AMBIGUOUS

The user may restate the command, identifying which PARKER he means.

If the data being input is already recorded, the system so states:

DATA: LT. COL. PARKER'S SERIAL NUMBER
IS 96738332!

INFO ALREADY IN DATA

Although the current system does have a few more capabilities that have not been discussed above, a good deal more work needs to be done on inputting data. Additional interactive features are required and planned. Also needed are the ability to input more than one connective ring per input statement, the ability to input data while defining a word, and an improved ability for inputting characteristics that apply to each item in a given class. Also, only rudimentary capability now exists for changing data entries or deleting them from the data base.

VERBS

Grammar rules involving verbs are somewhat different from those described so far. The difference is that these rules do not check data structures imme-

diately. Rather, each contributes to the building of a *verb table*. A *verb table* specifies an event space and an event that is hypothesized to have occurred within that space. Finally, in a rule that forms an S-phrase, the validity of this hypothesis is tested against the data base by a set of programs called the "verb routine."^{*}

A verb is considered to specify an *event*, which is defined as the existence of a state or a change of state as indicated by a time-dependent three-link connective ring in the data base. (See Fig. 10.) Specifically, the verb is associated with the referent ring that is used as an attribute. Also, it concerns a particular aspect of the relationship—a beginning, continuing (existence) or ending of the relationship. For example, ARRIVING, VISITING, and DEPARTING are associated with a beginning, continuing, and ending, respectively, of a LOCATION.

The associated attribute, aspect, and tense of a verb are specified in its definition. This information, which is provided by the dictionary lookup as part of the initial verb phrase, forms part of the verb table.

A completed verb table and tree diagram are shown in Fig. 11 for the sentence:

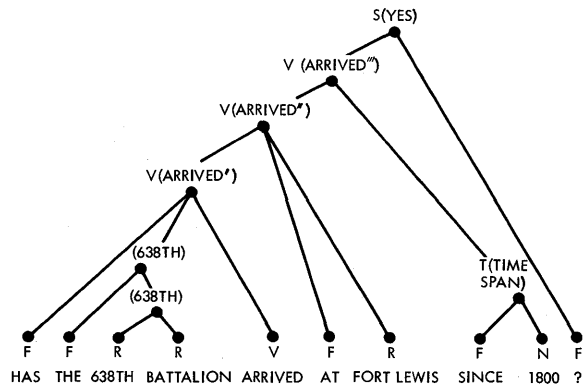
HAS THE 638TH BATTALION ARRIVED AT FORT LEWIS SINCE 1800?

Rules 5 and 11 apply as previously shown to reduce THE 638TH BATTALION to 638TH. Rule 51, HAS + R + V → V, then applies to HAS 638TH ARRIVED. This rule checks the compatibility of the auxiliary verb with the tense of the main verb, and places 638TH on the verb table as Subject. The output phrase in the tree diagram is labeled ARRIVED', with the prime indicating that the verb table has been extended or modified. Other rules place FORT LEWIS on the verb table as Value and set the time span as 18000 to 20500 (the implicit current time).

After the verb table has been completed (at the node labeled ARRIVED''), the "hypothesis testing" rule, V + ? → S, searches the data base for a BEGINNING LOCATION of the 638TH at FORT LEWIS between times 18000 and 20500. Since Fig. 10 shows an appropriate arrival (at 20000), the response to the sentence is YES.

The verb routine is also used in a variation of its

^{*} The verb routine programs were developed by Russell J. Abbott.¹²



SUBJECT	638TH
ATTRIBUTE	LOCATION
ASPECT	BEGIN
VALUE	FORT LEWIS
TENSE	PAST
TIME T1	1800
TIME T2	20500 (NOW)

Figure 11. Tree diagram and verb table.

role as a hypothesis tester to fill in gaps in the "completed" verb table. For example, consider the sentence

WHEN DID THE 638TH ARRIVE AT FORT LEWIS?

The verb table would not actually have a gap, since the phrase DID...ARRIVE sets the time span as past. However, in checking the data base for structures corresponding to past arrivals of the 638TH at FORT LEWIS, the verb routine fills in the specific times for which arrivals are recorded. Rule 308, WHEN + V + ? → S, therefore executes the verb routine, retrieves the list of times (or time spans, i.e., "between _____ and _____," in case of implied arrivals for which a definite time is unavailable) and prepares the list as the response to the sentence. Similarly, if the sentence did not specify the place of arrival, the verb routine would fill the gap with a list of places in which the 638TH had arrived. In more general terms, it fills in all appropriate values of the verb's associated attribute for the sentence subject. However, no rules of grammar have yet been written to take advantage of this type of verb table gap-filling.

Reference 12 describes the use of list processing "generators" which are used in conjunction with the verb routine for dealing with quantifiers. In addition

to quantifiers such as “any” and “all,” the concept of quantifiers/generators applies to “what” and “how many.”

In addition to its use in queries, the verb table is also convenient for inputting time-dependent data. Since it characterizes an event, a similar characterization in ring structures can be established if the verb table is specific enough. This use of the verb table was discussed earlier, in the section “Ring Structures and Data Input.”

More complex verb tables, which result from more complex sentences, are discussed in Reference 12. Additional grammar rules, both verb and non-verb, are also shown in Reference 12. The strategy for determining what rules are applied in what sequences is described in the following section.

THE LEVEL PARSER

The procedure used by the DEACON system to apply rules of grammar is a bottom-to-top, rewriting parser that produces, in parallel, all analyses of a sentence that the grammar allows. The unique feature of the parser is its use of level conventions to restrict redundant application of grammar rules.* This is an especially important consideration in a parser which was designed to use a context sensitive phrase structure grammar with discontinuous rules.

As indicated in the preceding examples, the parsing tree (Fig. 7) that represents an analysis of the sentence is built from the bottom up. The parser's ultimate goal is to apply enough grammar rules to collapse the sentence to a single sentential phrase (S). Analysis proceeds in steps from the sentence to (hopefully) the S-phrase goal. At each step in the parsing, the sentence is represented within the computer as a list of phrases, which is called a *subgoal*. Such a list made up exclusively of initial phrases is called an *initial subgoal*. Every subgoal is a representation of the sentence, but only the phrases in initial subgoals correspond directly to terms of the sentence. Other subgoals are a partially parsed representation of the sentence and contain at least one

*This particular DEACON parsing procedure is presented here because, as far as we know, it is unique in its use of levels. Kuno has suggested a modification which he believes would make the Level Parser more efficient, and Hays has suggested that perhaps the level parameter could be incorporated into the Cocke-Robinson parser, although the return on the effort to do so would probably not justify it. The idea for a level parser was conceived by Thompson; its implementation in the parser described here was by Gregory D. Gibbons.

intermediate phrase. This distinction between partially parsed subgoals and initial subgoals is ignored by the parsing procedure, but is useful in describing the parser.

The DEACON parser is a *rewriting* parser, which here means something more than a parser that employs rewriting rules of grammar. For, when a rule of grammar successfully applies to a sequence of phrases in a subgoal, the whole subgoal is rewritten.* The output phrase from the grammar rule is substituted for its constituents and the other phrases in the subgoal are copied. The new subgoal is added to a list of subgoals and is parsed along with them.

Thus, the list of subgoals represents all possible analyses of one† initial subgoal that have been developed up to a certain point. All potential analyses are developed in parallel. In contrast to a parsing strategy that follows a single analysis as far as possible and then backtracks to get other analyses (if any), the Level Parser carries all analyses along together and discards unproductive ones.

The Level Idea

The level idea is based on the standard representation of an analysis of a sentence as a parsing tree (Fig. 12). If only context-free rules of grammar are used, a level corresponds to the height of a phrase within the parsing tree. Phrases in an initial subgoal

* Although it is not necessary to rewrite the whole subgoal, the context phrases in context sensitive rules and the intervening phrases in discontinuous rules must be rewritten. A new parser which rewrites only what is necessary, but which does not use levels, has been implemented.

† There may be more than one initial subgoal for a sentence if terms of the sentence have alternative syntactic definitions. In such cases, each initial subgoal is parsed separately due to lack of core space.

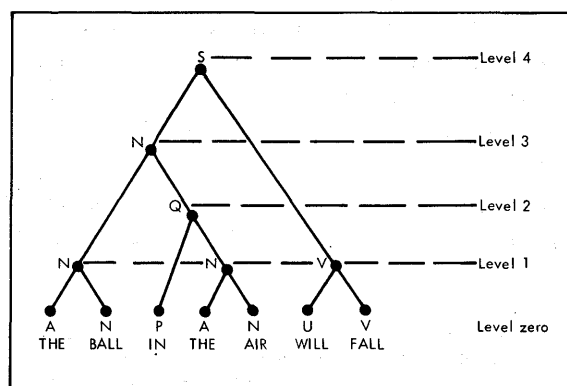


Figure 12. Levels in a context-free parsing tree.

are assigned level zero; they are at the bottom of the parsing tree. Any phrase produced by combining phrases is represented as a node which is higher in the parsing tree than the nodes which represent its constituents. A phrase which is produced by application of a grammar rule to phrases in the initial subgoal is, for example, assigned a level of one. This intuitive idea of level was modified slightly so that context sensitive rules could be used. This modification is necessarily detailed and is described in conjunction with the two simplified parsing examples below.

Using the level idea, the parser first combines phrases at the bottom of the parsing tree. When all that can be done at this lowest level has been done, the initial subgoal is discarded and the parser considers the next higher level, or level one. When all subgoals that are at this level have been used, they are discarded and the next higher level is considered. Finally, when the subgoals at some level are discarded, there is nothing left to parse.

The Level Conventions

The level conventions and the general path of the parser work in conjunction to reduce* redundant application of grammar rules to the same sequence of phrases. The level conventions are given below; the path of the parser is described by fiat in the two examples which follow the level convention description.

The parsing routine keeps a *master level* which defines the level within the parsing tree where the parser is currently working. Because the parsing routine begins at the bottom of the parsing tree, the master level is initially zero.

Each phrase in every subgoal is assigned a level. The level for initial phrases is zero. Any phrase output by a grammar rule is assigned a level equal to one plus the master level. Unaffected phrases in the subgoal are copied with whatever level they happen to have.

As the parser systematically matches phrases in the subgoals to rules, the following level restrictions must be met:

* All such redundancy cannot be eliminated because of context sensitive rules and because alternative initial subgoals are parsed separately. A phrase history, which contains rule outcomes for each sentence, eliminates actual re-application of rules.

Table I. Subgoals produced by the Level Parser in a simplified context-free example.

Master level = 0	Rules
Initial Subgoal $A_0 B_0 C_0$	1) $A + B \rightarrow E$ 2) $C \rightarrow F$ 3) $E + F \rightarrow S$
After application of Rule 1 in the initial subgoal	
Initial Subgoal $A_0 B_0 C_0$	Subgoal 1 $E_1 C_0$
After application of Rule 2 in subgoal 1	
Initial Subgoal $A_0 B_0 C_0$	Subgoal 1 $E_1 C_0$ Subgoal 2 $E_1 F_1$
After application of Rule 2 in the initial subgoal	
Initial Subgoal $A_0 B_0 C_0$	Subgoal 1 $E_1 C_0$ Subgoal 2 $E_1 F_1$ Subgoal 3 $A_0 B_0 F_1$
Master level = 1	
After application of Rule 3 in subgoal 2	
Subgoal 1 $E_1 C_0$	Subgoal 2 $E_1 F_1$ Subgoal 3 $A_0 B_0 F_1$ Subgoal 4 S_2

1. No rule of grammar will be applied if any of its input phrases has a level greater than the current master level.
2. No rule of grammar will be applied to a sequence of phrases unless at least one phrase in the sequence has a level equal to the master level.

Restriction 1 postpones the processing of recently produced phrases until the parsing routine begins work at the next higher level. Restriction 2 insures that the phrases at a lower level, which have already been processed, will not be done again.

Examples

Table I shows an abstract example where the level conventions are used to reduce redundant parsing. (Unfortunately, an interesting example is too lengthy for presentation at this time.) The example also gives a more detailed explanation of the order of parsing.

The initial subgoal is made up of three phrases with categories A, B, and C, respectively. Each initial phrase is given level zero (indicated by subscripts). The master level is initially zero.

Parsing begins at the left on phrase A in the initial subgoal. Rule 1 matches phrases A and B and after successful application, the initial subgoal is rewritten incorporating the new phrase E, which is assigned a level one.

The parser moves *down* the first column to the phrase E in subgoal 1. Consideration of this phrase is aborted because the level of E is greater than the master level. Parsing of phrase E is postponed until the master level is raised to one.

The parser moves to the top of the next right column and finds that no rules apply to phrase B. Moving down the second column, Rule 2 applies to phrase C in subgoal 1. New subgoal 2 is added to the bottom of the subgoal list. Phrase F in subgoal 2 is out of range and Rule 2 applies to phrase C in the initial subgoal to produce subgoal 3. Continuing, the parser finds that there are no 3rd phrases in subgoals 1 and 2 and that phrase F in subgoal 3 is out of range.

The parser has completed its first left-to-right sweep. Everything that can be done strictly at the bottom of the parsing tree has been done. The master level is raised to one. Now each sequence of phrases to which a rule can apply must have at least one phrase at level one. Because no sequence of phrases in the initial subgoal can meet this requirement, the initial subgoal is *discarded*. Whenever the master level is raised, subgoals containing only sequences of phrases that are now out of range are discarded.

Beginning at the top left and moving in the established pattern, the parser finds that no rule applies to the sequence E C in subgoal 1. Rule 3 applies to phrases E F in subgoal 2, producing subgoal 4.*

Rule 1 would normally apply to the string A B in subgoal 3. But this has already been done. Rule 1 was applied to this particular string in the initial subgoal. If it were applied again, the resulting subgoal would be exactly like subgoal 2. Level restriction 2 (at least one phrase level equal to the master level) prevents this. Since neither phrase A nor phrase B in subgoal 3 has level one, the sequence is not re-parsed. Restriction 2 also applies to phrase C in subgoal 1 and the parsing of this example is completed.

* In actual practice, phrases with category S are not rewritten into the subgoal list, but are set aside on a special output list.

Table II. What level should be assigned a phrase which has been used as context?

Master level = 0	Rules
Initial Subgoal: $V_0 W_0 X_0$	4) $V + W \rightarrow Y$ 5) $W + X \rightarrow W + Z$ 6) $Y + Z \rightarrow S$
After application of Rule 4 in the initial subgoal	
Initial Subgoal $V_0 W_0 X_0$	
Subgoal 1 $Y_1 X_0$	
After application of Rule 5 in the initial subgoal	
Initial Subgoal $V_0 W_0 X_0$	
Subgoal 1 $Y_1 X_0$	
Subgoal 2 $V_0 W_2 Z_1$	

The general problem in allowing context sensitive rules is that of having the context available when it is needed. Because the DEACON parsing method rewrites entire subgoals, the context is always available when needed. The problem, instead, becomes how to insure that the context is properly parsed *after* it has been used as context. Consider the example in Table II.

Following the parsing procedure outlined in the previous example, Rule 4 and Rule 5 are applied to the initial subgoal V W X, resulting respectively in subgoals 1 and 2. At the end of the first left-to-right sweep, there are three subgoals, namely the initial subgoal, subgoal 1 and subgoal 2. The master level is raised to one and the initial subgoal is discarded. Subgoal 1 is obviously a blind alley; there is no way to parse it to completion because in creating the Y-phrase, the context W which is required to parse the X-phrase was used up.

Subgoal 2, on the other hand, represents the case where the context W has been properly used to parse the X-phrase into a Z-phrase. But now what can be done with the phrase sequence V W? If the intuitive level idea is followed strictly, it is obvious that phrases V and W are still at the bottom of the parsing tree and hence each might have a level zero. But because the master level has been raised to one, level restriction 2 will prevent the re-application of Rule 4 to the V W sequence. In this case, no further rules could be applied to subgoals 1 and 2. When the master level is raised to 2, all subgoals that do not contain at least one phrase with level 2 are discarded. Subgoals 1 and 2 are discarded and there

Table III Parsing continued after appropriate assignment of level to context phrase.

Master level = 1	
After application of Rule 4 to subgoal 2	
Subgoal 1	$Y_1 X_0$
Subgoal 2	$V_0 W_1 Z_1$
Subgoal 3	$Y_2 Z_1$

is nothing left to parse. No analysis of the sentence V W X could be found.

To correct this undesirable state of affairs, the level idea is pragmatically extended. Any phrase "output" by a grammar rule is assigned a level equal to one plus the master level. Phrases "output" from a grammar rule are the single new phrase and any phrases which should be retained as signalled by the grammar rule. Assignment of level to the new phrase corresponds with the intuitive level idea. The phrases which are to be retained are the context phrases. So any phrase used as context is also assigned the new level and not just rewritten with whatever level it happened to have.

A review of Table II with this new level convention in mind will show that the W phrase in subgoal 2 is assigned level 1. Then the sequence V W meets the requirement of having at least one phrase with level equal to the master level and Rule 4 may be applied. (The phrase history list is consulted to obtain this previous result directly.) The problem of what to do with context phrases is resolved, as shown in Table III.

At the end of the left-to-right sweep, subgoals 1 and 2 are discarded as above, but this time there is a subgoal left to parse. Application of Rule 6 to subgoal 3 results in the desired completion of parsing.

This slight modification to the level conventions changes the nature of the idea of level. Instead of "height within the parsing tree," the levels, therefore, no longer say something about the phrase to which they are attached. Instead, they indicate something about the subgoal containing the phrase to which they are attached.

HARDWARE CONFIGURATION

The current experimental DEACON system is implemented on a 16K GE-225 general purpose digital computer with a 20-bit word length and an 18

microsecond cycle time. A random access disc unit of six million words is used for secondary storage. Model 33 or Model 35 teletypes connect to the mainframe through a DATANET-15 interface unit, which allows interactive use but no time-sharing. The hardware configuration also includes a card reader, a punch, a printer, and six tape units. These devices are used for service jobs such as building the system, assembling programs, and taking dumps.

CONCLUSIONS

The DEACON system is in its second phase of development as a management information system featuring English language data input and query capabilities. The first phase, the DEACON Breadboard,¹³ proved the feasibility of using interpretation rules for relating English statements to computer-stored data structures. The data base was pre-stored in list structure form as a static part of the system. No time-dependent (i.e., historical or projected) data was permitted. Queries could be given in English, on punched cards, and system responses were by printer. Peripheral memory was magnetic tape, and a considerable part of the development effort consisted of devising techniques for using list processing in conjunction with peripheral memory.

The present system includes the following major advances over the Breadboard system: (1) the provision of direct access to the computer via teletype, permitting interactive use for developing the system and experimenting with it; (2) the ability to input data from the teletype in English; (3) the use of ring-type data structures, which have proven richer than the earlier list structures; (4) the use of disc peripheral memory rather than tape; (5) the use of generally more efficient processing routines such as the parser and the dictionary and data handling techniques; (6) the incorporation of time-dependent data; and (7) the use of a more generalized method for handling verbs.

Future work will concentrate on adding new features as well as extending and improving the efficiency of current capabilities. Perhaps the most significant new feature needed is the ability to define vocabulary terms in English, using previously defined terms. The importance of this capability is discussed in reference 9 as "recursive definition and the recursive behavior of structural modification." Some initial work is being done on this and on algebraic grammar rules. Other major areas of planned grammar

extension include pronouns (with some cross-sentence referencing), negation, and clausal modification. The grammar currently consists of some 250 rules. Various techniques are under consideration to make it easier to write grammar rules.

Data structures are receiving much attention as a promising area for increasing the semantic capability of the system, and for improving processing efficiency. The processing systems such as the parser and dictionary and data handling techniques are also being improved apart from the data structures work.

Sentence processing time varies normally from under a minute to three or four minutes, but has gone as high as 17 minutes. System improvements as mentioned above, greater use of machine-language programming, and use of a larger, faster machine should much more than offset the slowing tendencies to be expected from enlarging the grammar and data base.

Our experience with the DEACON system has convinced us of the soundness of its theoretical basis. Conclusive proof may await the test of a prototype operating in a live, rapidly changing context. However, we look forward to such an operation as a source of guidance for continuing development of the DEACON system.

APPENDIX: SAMPLE RULES AND SENTENCES

Some R-phrase modification rules have been illustrated, such as Rule 5, $R_1 + R_2 \rightarrow R_1$, and Rule 8, $R_1 + OF + R_2 \rightarrow R_3$. There are several other rules that take two R-phrases or two R-phrases and a preposition as input, and produce an R-phrase as output. For example, Rule 1 is $R_1 + R_2 \rightarrow R_2$ in which the output R-phrase has the same data referent as the second input R-phrase. It is needed for strings like

$$\begin{array}{ccccc} R_1 & + & R_2 & \rightarrow & R_2 \\ LT. COL. & & PARKER & \rightarrow & PARKER \end{array}$$

This rule checks for existence of an appropriate connective ring between the referent rings of LT. COL. and PARKER. If satisfied, it uses the referent ring of PARKER as the referent ring of the output phrase. That is, it verifies that PARKER is a LT. COL. before accepting the string LT. COL. PARKER as a syntactic unit. Or, assuming that there is more than one PARKER but only one is a LT. COL., this rule selects the appropriate referent ring

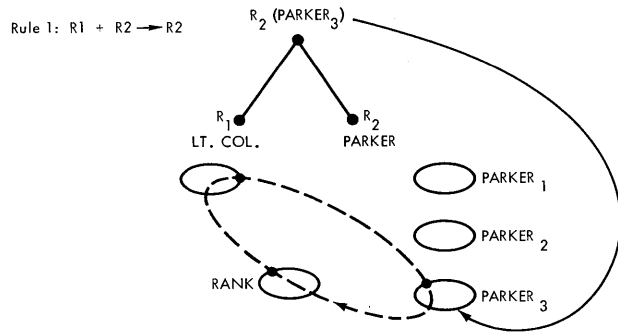


Figure 13. Example of resolution of multiple-data-referent ambiguity by data checking in a linguistic modification rule.

for the output phrase. As shown in Figure 13, the name PARKER is ambiguous in the input phrase because it leads to three distinct referent rings.

In this case, the data check resolved the ambiguity. Obviously, if the data base had been different, some or all of the ambiguity might have remained; or, if no PARKER were a LT. COL., the output R-phrase would be marked VACUOUS DESCRIPTION. If no successful analysis results from parsing LT. COL. and PARKER differently (conceivably in something like "WHO WAS THE LT. COL. [THAT] PARKER SUCCEEDED?"), the VACUOUS DESCRIPTION (LT. COL. PARKER) is typed out as a clue to the failure to answer the query.

Since Rule 1 and Rule 5 each require two adjacent R-phrases as constituents, an attempt is made to apply both rules to LT. COL. PARKER (in alternative parsings). If successful, Rule 1 interprets the phrase as PARKER, and Rule 5 interprets it as LT. COL. However, the connective ring shown in Figure 13 does not satisfy the requirements of Rule 5, so this possible ambiguity is quickly resolved. Both rules do successfully apply to the string 638TH BATTALION as shown in Figure 14. But since the referent ring of BATTALION does not link to that of COMMANDER, the parsing in which Rule 1 was used is aborted at the next step. If such ambiguities are not resolved, and result in different answers to a query, the alternative answers are typed out along with their respective parsings.

Rule 2, $R_1 + R_2 \rightarrow R_3$, introduces a new type of output phrase referent ring—a "scratch" ring created by the rule and linked by connective ring to appro-

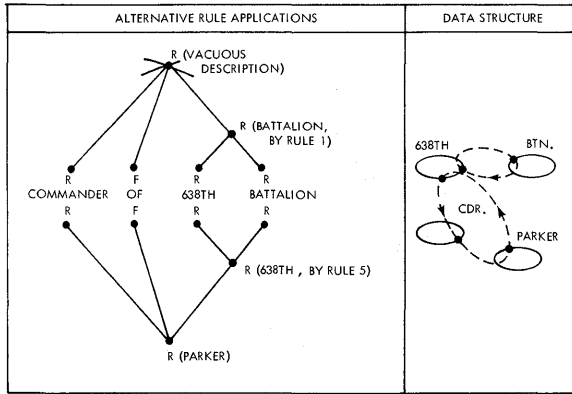


Figure 14. Ambiguous interpretation (by two alternative rules) of a given phrase resolved by a data check at a higher level phrase.

appropriate referent rings in the permanent data base. The application of this rule to the string ENGINEER BATTALIONS causes such a scratch ring to be formed. (Fig. 15.) This rule also illustrates an interesting aspect of ambiguity. Although each constituent R-phrase is unambiguous, the combination may be ambiguous because their data referents are interconnected via alternative paths. For example, if some BATTALIONS are HEADQUARTERED at FORT IRWIN and others are temporarily LOCATED there but HEADQUARTERED elsewhere, an ambiguous response would result from the statement

LIST FORT IRWIN BATTALIONS.
 BATTALION (HEADQUARTERS) FORT IRWIN
 522ND
 436TH
 593RD
 BATTALION (LOCATION) FORT IRWIN
 638TH
 94TH
 523RD
 117TH

Two separate scratch rings are created in this example—one for each interpretation.

There are also prepositional forms of these rules:

Rule 8: $R_1 + OF + R_2 \rightarrow R_3$
 Example: COMMANDER OF 638TH \rightarrow
 PARKER

Rule 31: $R_1 + OF + R_2 \rightarrow R_1$
 Example: PARKER OF 638TH \rightarrow
 PARKER

Rule 35: $R_1 + OF + R_2 \rightarrow R_3$
 Example: BATTALIONS OF FORT IRWIN \rightarrow FORT IRWIN BATTALIONS
 (R_3 refers to two scratch rings as in earlier example.)

Rule 36: $R_1 + IN + R_2 \rightarrow R_3$
 Example: BATTALIONS IN FORT IRWIN \rightarrow FORT IRWIN BATTALIONS
 ("IN" eliminates the location/headquarters ambiguity.)

The dual application of Rule 8 and Rule 31 to "R OF R" expressions produces ambiguity that tends to remain unresolved in such questions as WHO IS COMMANDER OF 638TH? With both rules applying, the response would be

(₃₀₅ WHO IS (₈ COMMANDER OF 638TH) ?)
 PARKER

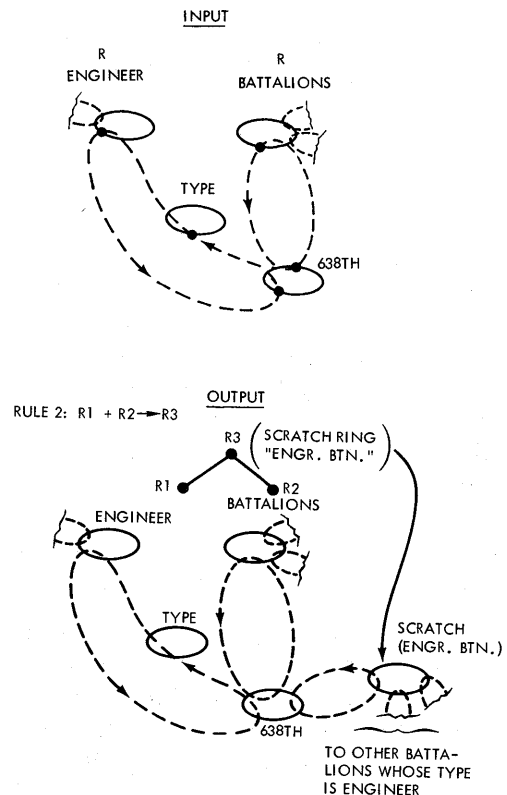


Figure 15. Creation of a scratch block, linked by connective rings to referent rings in the permanent data base, during application of a grammar rule in analyzing a query.

(₃₀₅ WHO IS (₃₁ COMMANDER OF
638TH ?)
COMMANDER

Actually, the restrictiveness of the pronoun "WHO" eliminates the second output in this example unless the system has been told that the word COMMANDER itself is in the range of the pronoun WHO.

In practice, a more general ambiguity resolver eliminates Rule 31's application sooner. Under "Ring Structures and Data Input," it was stated that input statements of the form "R₁ OF R₂ IS R₃" caused the data referent of R₁ to be placed on a special OF referent ring as a reminder that it may be used attributively. Rules 8 and 31 compare their R₁ data referents with the OF ring. Rule 8 applies if R₁ is on the OF ring, and fails if it is not; Rule 31 applies if R₁ is not on the OF ring, and fails if it is. These

complementary checks, which resolve the PARKER/COMMANDER type ambiguity, amount to adding the part of speech "ATTRIBUTE".

These few rules have been presented in some detail to give the "flavor" of the problems involved in developing the DEACON grammar. These and similar rules used recursively and in combination with the verb rules, data input rules, and sentence-forming rules such as

Rule 301: LIST + R + : → S

Rule 305: WHAT IS
WHAT ARE + R + ? → S

Rule 307: WHERE IS + R → S,

successfully analyze a wide range of sentences involving attributive relationships. Some sample sentences that the system now accepts are shown with their responses in Fig. 16.

1. List all engineer battalions.
425 TH
638 TH
94 TH
523 RD
682 ND
496 TH
117 TH
2. List the commanders of engineer battalions.
J.C. Irvin
D.J. James
J.M. Parker
S.V. Olsen
T.C. Allen
I.S. Parker
J.C. Conway
3. Where is Parker?
Ft. Lewis
Ft. Irwin
[Ambiguous - 2 Parkers]
4. Where is Lt. Col. Parker?
Ft. Lewis
[Only one is a Lt. Col.]
5. When will Lt. Col. Parker leave Ft. Lewis?
22000
[Times are currently treated as relative numbers on a "time line."]
6. It is now 23000!
OK
[There is now no system clock yet. The user must set "now" explicitly or it is automatically "0."]
7. Where is Lt. Col. Parker now?
Presidio
8. What is the date of rank of the commander of the 638 TH Battalion of Fort Lewis?
11600
9. What is Olsen's date of rank?
12900
10. Does Parker's date of rank precede Olsen's date of rank?
Yes
11. What staff colleges has Lt. Col. Parker attended?
Command General School
Infantry School
12. Has Lt. Col. Parker arrived at the Presidio from Ft. Lewis?
Yes
13. What is the readiness of Lt. Col. Parker's Battalion?
Orange
14. Data: The 425 TH will leave Ft. Lewis at 21950!
OK
15. Data: The 638 TH will arrive at Ft. Lewis at 20000!
OK
16. Is the 638 TH scheduled to arrive at Ft. Lewis before the 425 TH leaves Ft. Lewis?
Yes
17. How many combat engineer companies have trained at Ft. Irwin this year?
6
18. How many Sixth Army installations are there?
38
19. What is the distance from the 638 TH Battalion to San Diego?
61
[No units are currently stored. Forts and cities are located on a square world of xy coordinates, so "distance" is the "square root of the sum of the squares..."]

Figure 16. Sample sentences.

ACKNOWLEDGMENTS

Our greatest debt is to Dr. Frederick B. Thompson, who is clearly the author of DEACON, if not of this paper. Mr. John W. Gwynn and Mr. Gregory D. Gibbons contributed to the Breadboard and the experimental system. Mrs. Jacque Suzanne Pruett was a valuable member of the Breadboard team, and Mr. Russell J. Abbott, Mr. Robert L. Price, and Mrs. Marillyn Popkin have contributed to the experimental system since its beginning. Dr. John C. Fisher adeptly pulled together the enabling resources for the experimental system. The project is under the management of Dr. Herbert R. J. Grosch.

REFERENCES

1. S. Kuno and A. G. Oettinger, "Multiple Path Syntactic Analyzer," *Information Processing 1962*, North Holland, Amsterdam, 1963, pp. 306-312.
2. F. B. Thompson, "English for the Computer," *AFIPS, Volume 29, Proceedings of the Fall Joint Computer Conference, 1966*. (This volume)
3. B. F. Green, et al, "Baseball: An Automatic Question Answerer," *Computers and Thought*, McGraw-Hill Inc., New York, 1963.
4. R. K. Lindsay, "Inferential Memory as the Basis of Machines Which Understand Natural Language," *Computers and Thought*, McGraw-Hill Inc., New York, 1963.
5. R. F. Simmons, "Storage and Retrieval of Aspects of Meaning in Directed Graph Structures," SP 1975/001/02, System Development Corp., Santa Monica, 1965.
6. I. E. Sutherland, "Sketchpad: A Man-Made Graphical Communication System," *AFIPS, Volume 23, 1963 Spring Joint Computer Conference*, Spartan Books, Washington, D.C., pp. 329-346.
7. F. B. Thompson and R. W. Callan, "Concept for the Navy Operational Control Complex of the Future," (RM 62 TMP-49-1), General Electric, TEMPO, Santa Barbara, Calif., 1962.
8. F. B. Thompson, "The Semantic Interface in Man-Machine Communication," (RM 64 TMP-35), TEMPO, 1963.
9. ———, "Application and Implementation of DEACON-Type Systems," (RM 64 TMP-11), TEMPO, 1964.
10. ———, "Design Fundamentals of Military Information Systems," *Military Information Systems*, 1964.
11. "Information Systems Research," Project DEACON, (RM 65 TMP-69), TEMPO, 1965.
12. "Phrase Structure Oriented Targeting Query Language," (RM 65 TMP-64), TEMPO, 1965.
13. F. B. Thompson, et al, "DEACON Breadboard Summary," (RM 64 TMP-9), TEMPO, 1964.

COMPUTER ASSISTED INTERROGATION

Charles T. Meadow and Douglas W. Waugh

*IBM Corporation
Bethesda, Maryland*

INTRODUCTION

Summary

Computer Assisted Interrogation (CAINT) is a system of computer programs for use in man-machine communications. Its principal function is to enable a computer to elicit information from a man by interrogating him—asking him a program of questions where the program follows a logical course depending both on information available before the interrogation started and on that gained during the interrogation. The information acquired is intended to be put to immediate practical use, in updating a data base, generating reports, or driving other interrogations.

During the course of an interrogation, the interrogee will be given information as well as asked questions, and he may ask his own questions, as well as provide answers. Thus, a CAINTE interrogation is truly conversational, with information and questions flowing in both directions. The conversation, particularly in the machine-to-man direction, is somewhat stereotyped, the machine's versatility being limited by a repertoire of generalized, fragmented statements which are particularized and assembled for use as needed. The conversational range of the computer, then, depends upon a system user's versatility in designing these statements—a process somewhat akin to computer programming. In its present state, CAINTE imposes few restrictions on man-to-machine

communication although it is unable to perform any content analysis on a natural language response. We feel that CAINTE achieves one of the aims of man-computer symbiosis, as defined by J. C. R. Licklider,¹ "to enable men and computers to cooperate in making decisions and controlling complex situations without inflexible dependence on predetermined programs."

CAINT is applicable to a broad class of information acquisition problems. It is being developed in conjunction with a project whose aim is computer program documentation. Here, guided by the structure of the program itself, and design information previously contributed by the programmer and others, CAINTE interrogates a programmer about his own program and elicits a detailed, up-to-date, program description. It can also be used for other highly structured reporting activities and for advanced educational programs.

The Nature of the Conversation

CAINT superficially resembles Computer Assisted Instruction, CAI.² In CAI, information is presented to a student in short paragraphs, or *frames*: A question is asked about the information, the student answers the question, and the system responds to a student's answer. The student's answer controls branching within the instructional program, which can then produce a comment on the answer, change the

planned sequence to review the material if necessary, ask the student to try again, or go on to new material. In CAI, however, student responses must be anticipated. To oversimplify—the question “When did Columbus discover America?” anticipates the correct answer “1492.” The system may accept “fourteen-hundred and ninety-two” but might not recognize, hence might reject, “late 15th century.”* CAINT sometimes anticipates answers, but serves its greatest purpose when it asks questions, the answers to which are not known to anyone but the responder—information entirely new to the system. CAINT can also vary its question-asking routine by using information obtained from any prior response or any items in a data base whose information originates outside CAINT, perhaps in another information system. The conversational capabilities of CAI, then, are a subset of CAINT’s functions.

We can more formally define CAINT as a three way communications system. The communicating elements are: a data base; a man, or group of men, engaged in some activity; and an executive. The initiative lies with the executive, who is the person who contributes an executive program and the set of general statements. The executive program directs the system how to respond to different stimuli or conditions. It may specify, for example, that upon receipt of a change in item X of the data base, a certain set of statements is to be issued to the human responder, the exact form of these statements depending upon the particular values of item X before and after the change, and on other data base items. Responses to these statements by the responder may further modify the program of statements. Thus, each of the three communicating elements has some control over the conversation, with a sort of policy control exercised through the executive program.

CAINT permits an item of information arriving in a data base to be presented to a man, and enables the system to ask a set of questions about the datum. These are tailored to that datum in the context of the overall data base as it stands at the moment. In other words, different questions might be asked, in response to the same stimulus, given a different data base

* IBM’s Computer Assisted Instruction system enables a course writer to anticipate any of these answers and treat them as correct. He can even accept any answer which contains a given set of key words as “fourteen,” “ninety,” “two,” which would recognize “fourteen-ninety-two” as correct. However, this obviously requires both more computer time and more instructor time, and could be avoided by requiring dates to be entered as numbers.

environment. For example, in a management reporting system, a conversation between CAINT and a programming group leader might be, in part, as follows:

<u>Machine Statements</u>	<u>User Responses</u>
Your responsibility is:	1 DECEMBER 1966
Tracking Program. Estimate your completion date.	

The machine then responds to this estimate depending upon the history of the original assignment and responses to this question when asked in previous interrogations. Possibilities include:

Your estimate includes adherence to original schedule.	(NO RESPONSE REQUIRED)
Your last estimate indicated 1 October 1966. What is the reason for the delay?	DESIGN CHANGE PROMULGATED IN PROJECT MEMORANDUM NO. 123 INCREASED COMPLEXITY OF THE PROBLEM.

The term *data base* applies to files stored in a computer system accessible to CAINT. It is important to note that data base information can come from external sources or from responses to CAINT interrogations. We will occasionally use the term to refer solely to data selected from other than interrogation sources, but we do this to stress that CAINT can be made susceptible to external control, not to imply that responses cannot enter the data base.

The term *responder* will generally be used to refer to a single individual. All our current experimental work centers around individual responders. However, there is no conceptual bar to thinking of the responder as a group being independently interrogated on the same subject, a team, each member of which performs a different function contributing to a common goal, or another data processing system. For example, a group of managers might work together to plan a schedule and organization for a new project.

CAINT is a practical information acquisition tool. The result of an interrogation is to add information to a file. The CAINT system has the capability to compile documentary reports from the data base, using the same logical capability to select and se-

quence interrogation frames. This use of CAIN T reduces the cost and time required to produce the right information to meet the needs of the moment.

A Survey of Applications

We discuss here four possible applications of CAIN T. In a later section (Program Logic), we expand one of these examples to illustrate the logic of CAIN T programs.

1. Programming documentation. A computer program is a complex structure, with many interlocking threads, with data sets or files shared by some parts of a program, not shared by others, data names or labels meaning the same thing in one part, different things in others. The problems inherent in changing someone else's program are notorious, largely due to inadequate documentation. Our objective is not only to produce documentation good enough to enable a programmer to change another's program, but to have a new version as well-documented as the original, so a third party can change the joint work of the first two. We find that programmers easily forget details of their programs (witness the relearning process after each debug run) and need to be taught much of their own program, in order to document it successfully.

Given a completed computer program to be documented, we perform a preliminary analysis of it, breaking it down into segments and providing such information as, for each segment, the possible successor segments and the numbers and types of statements or instructions used. This information is stored in the data base. The executive program uses the successor table to follow threads in the program, considering segments in the order in which they might actually be executed. By examining detailed information about the statements in each segment, the executive program decides what specific questions to ask; whether, for example, to ask for an explanation of the meaning of a file of input data, or of the limit on the index of a DO statement. The programmer is then asked a series of informed questions about his own program, interspersed among statements reminding him about the program's structure, use of data, etc. He responds mainly to requests for explanation or amplification of data already found in the data base. Different using organizations could provide for different forms of documentation by varying the executive program or the preliminary

analysis that creates the bulk of the data base. The responding programmer will contribute some data base items and could be given control over the path to be followed whenever the choice is arbitrary, such as at a conditional transfer. Thus, the executive programmer, the responding programmer, and the program to be documented each exert some control over the final document and the sequence of steps taken to produce it.

2. Management information systems are another possible CAIN T application. Too often, these are merely storage and retrieval systems for fixed reports or questionnaires. Questionnaires on complex subjects tend to become cumbersome and confusing because there is so much variation in how questions can be answered, and, sometimes, each possible answer to a question generates a different set of ensuing questions.

Actually, out of a large number of possible questions concerned with, say, progress reporting, only a few may be applicable to any particular manager. We would like to see these, and only these, asked. This will reduce the time spent on reports, and, even more significantly, will change the report from a highly stereotyped, virtually meaningless, document into one whose relevance is obvious to the interrogee, and to which more attention and respect will then be paid by him.

The report that any individual manager (or individual engineer, programmer, salesman, etc.) is asked to file, then, consists of his answers to a series of informed questions which enable him to concentrate only on areas of significance, to explain when necessary, and to do this with a minimum of effort. This gives the project manager or reviewing authority specific answers to questions that are only asked because there is something in the files that indicates that they should be asked, rather than stereotyped answers to stereotyped questions.

3. Military intelligence is an area in which the importance of the information required depends not only upon who is answering the questions, but what the current politico-military situation is. For example, one would not normally ask a political intelligence analyst the tactical significance of a new type armored vehicle. An aerial photograph interpreter follows a certain general regimen in analyzing photos, but the specific facts needed by the military commander to whom he reports can vary. Topographic

information may be of first importance when an army is moving rapidly into unfamiliar terrain, but not so in a static situation. Perhaps an on-the-spot observation by a patrol may create an immediate need to reexamine photos of a particular area, to verify some reported item. The generation of the appropriate questions for the interpreter can be handled by CAINT so long as the implications of the arrival of the patrol's report in the data base can be programmed. We elaborate on this application in the section on program logic.

4. Swets and Feurzeig,³ describing a possible application of computer-aided instruction, give an illustration of a teaching system for medical students. Their example differs from the conventional programmed instruction course in that the machine, rather than volunteering its store of information, often waits for a student input and then responds with tutorial information specific to that input. Thus, to the student, the system appears to know more about the patient, whose condition is the subject of the course, than is given explicitly by the course. The machine's responses to students' inputs, then, appear to be based on data base information as well as the student answers. CAINT could carry this a step further and allow the use of a real data base in providing this type instruction. Furthermore, it could perform a double service by interrogating the doctors, nurses, and technicians in charge of the patient to assist them in making their reports, thus acquiring information on the one hand, and teaching it on the other.

SYSTEM DESCRIPTION

Overall System Description

We define an *object system* as that which is described by the data base, and which is the object of most of the conversation between man and machine. The object systems we have illustrated include a computer program, the management control system for a project or other multi-person activity, an aerial photograph (or in a larger sense, the entire system of intelligence available to an armed force), the status of a hospital patient and various related hospital activities.

In addition to conversing about the substance of some object to be documented, CAINT requires two-way communication on what amounts to administrative subjects. We differentiate between *substantive* and *service messages* transmitted in either direction between the computer and the system user. Sub-

stantive messages either give the user information from the data base which has some external meaning to the object system, or they are responses to questions and are to be added to the data base as new information or commentary on existing information. A service message, on the other hand, might ask the narrator which of two paths he wishes to follow in an interrogation. The answer to this question is also a service message. Neither this question nor the answer are descriptive of the object system. The distinction is not always clear-cut. A message asking which path to take also conveys to the narrator the substantive information that there are two paths, but his answer adds no substantive information to the data base.

Although not clear-cut, the distinction is important. Even in early simulations, we have found that there is a high percentage of service message flow during an interrogation, and these messages have relatively little intellectual content. Hence, the narrator reads them quickly and comprehends them easily. The twofold nature of information transmitted introduces a human engineering problem of presentation of service frames, for the user can be easily bored or distracted by slow presentation of uninteresting service information. Substantive information requires thought while reading or composing. Hence, time delays imposed by slow transmission are not critical. Cathode ray tubes or other high-speed displays may be indicated for the service messages, while typewriter speeds have so far been acceptable for the substantive messages.

The system user does not need elaborate training to operate CAINT. The system is designed to provide built-in, computer assisted training which, while specific to the particular application, is relatively easy to supply and would presumably be employed in each application. In our program documentation application, there are two programs of instruction for system users. One explains the mechanics of the use of CAINT, how to select and change modes, and what these modes are. A second, much longer, course teaches our approach to program documentation and some of the special jargon we have employed, and takes the neophyte user through a sample interrogation and report production.

Finally, we reiterate, the total system is one in which messages flow as follows:

Data base changes. These can enter directly through a storage and retrieval system into the data base.

Information statements and questions— from the CAINT system to the users. Statements and questions can inform users, on demand, about the content of and activities in the data base, can instruct them on information they need to operate the system or to branch within an interrogation or instruction program.

Information statements and questions— from man to the machine. Again these can be about the object system, can be questions about procedures involved in CAINT, or can be answers to either object-system related questions or procedural questions.

Tabular and narrative reports. These are generated by the system and can then be transmitted along with any other messages to any output devices available. Hence, CAINT could be used for transmitting nonverbal messages to system components in control system applications, as well as for providing conventional documentation.

Operating Modes and Activities

CAINT operates in three basic modes: control, conversation, and utility. In any mode there may be several activities permitted. An *activity* is a specific function performed by a user. The system is in a given *mode* if a particular set of computer programs is loaded. The *control mode* provides entry into the system, indoctrination in system mechanics, and enables the user to switch among modes and activities. The *conversational mode* enables the user to select among a set of production activities: *narration*, *report compilation*, *editing*, and *instruction*.* The *utility mode* covers the storage and retrieval activities.

Control Mode. A user performing, say, narration can switch to retrieval by invoking the control mode and specifying the new activity he wishes to shift to. He can then shift to the instruction activity, then back to narration, at the place where he left off, and complete his original task. Figure 1 shows the relationship among the modes and activities. Note that,

* We use the term *instruction* to mean teaching a user about information in the data base, and *indoctrination* to mean teaching the use of CAINT.

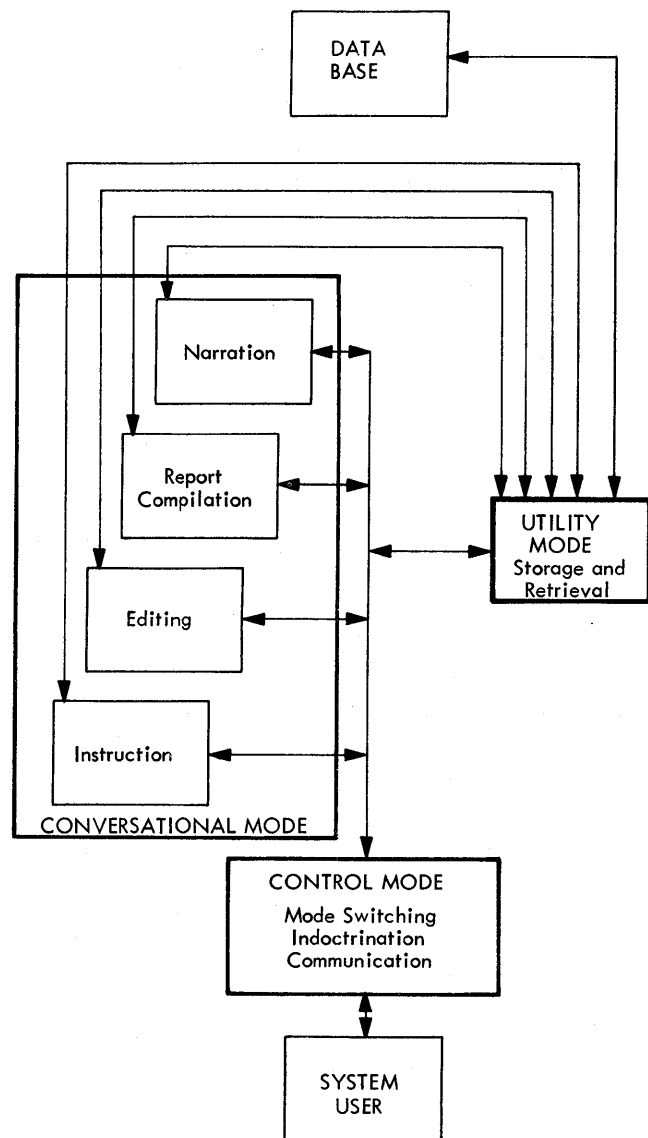


Figure 1. Relationship among modes and activities.

from any mode or activity, to make a change, the control mode is invoked and the change made from there.

In order to initiate an activity, the user first enters the control mode. This mode receives all user requests for activity selection or processing, and takes appropriate action. All user activities are handled by computer-initiated acquisition of user commands. That is, rather than requiring the user to enter control cards or otherwise initialize the system for a particular application, CAINT elicits control information in much the same way as it gathers the user's substantive knowledge about the object system. An

important feature of the control mode is the incorporation of an *indoctrination* capability. Indoctrination is instruction in the capabilities and use of CAINTE.

Conversational Mode. The principal CAINTE functions are performed in the conversational mode. This is the mode in which messages are selected for transmission to the user, and his responses are acted upon. We shall show in the section on program logic how several conversational activities are actually handled by the same basic set of computer programs.

Narration, or, from the computer's point of view, *acquisition*, is the activity in which a man contributes his own substantive knowledge to the system data base. This is the key activity for which CAINTE was designed. This system produces a sequence of frames, some, but not necessarily all, being questions. The frames which are not questions are usually serving to inform the man about the contents of the data base preparatory to asking him some questions about them, "priming" him. The important questions call for new information—discrete facts or interpretations of existing items. Some of the information flow, in both directions, is service information concerned solely with sequencing—deciding what topics or frames to consider next.

We have found that the sequence in which information and question frames are introduced, in order to elicit information, is not necessarily the sequence in which a reader would like to review the information elicited. The *report compilation activity* enables a user to assemble his responses, together with some of the purely informational frames, prepared titles, transitional text, and data base items other than question responses, into a report organized for ease of reader comprehension. As we shall see, the report requires more computer programming than does conventional report generation where the objective is to selectively dump portions of a file.

Since the CAINTE user (or narrator or responder) answers a series of discrete questions which follow an overall approach dictated by the executive program, he would probably like to review and revise his copy before publication. The report compilation activity compiles a report designed to aid the reader. The narrator can use this for review and modification—the *edit activity*. Editing gives the user the chance to read his output, in a context different from that in which it was written, to indicate those portions he wishes to change, to make the change, and to update the data base as he does so. The narrator

flags the portion of the report he wishes to change,* CAINTE finds the set of frames that asked the question that led to this response, the set is recalled and asked again, new responses are stored in the system files, and changed report copy is issued. All this is done on-line, in the conversational mode.

Editing of computer-stored, digitized text has been accomplished by several different program systems.^{4, 5} These programs use much more elaborate editing commands than those used in CAINTE, but one factor makes the edit process in CAINTE strikingly different from conventional text editing. This is the fact that, in CAINTE, there are several integrated processes all making use of a single data base. Thus, a change to a report alters the data base and then can cause changes to future acquisition, instruction, and reporting activities.

Once the narrator has established a file of responses that are acceptable to him, on his review in the edit activity, this information can be made available to others in an instructional format. A person who wishes to learn, say, a program that has been documented through CAINTE engages in the *instruction activity*. Here, the narrator's responses and other data base items are embedded in instructional frames (using an executive program to do so) and then presented in a sequence guided by the executive program, but modifiable by the learner. This mode of instruction fits the definition of programmed instruction given by Lysaught and Williams,⁶ "arranging materials to be learned in a series of small steps designed to lead a student through self-instruction from what he knows to the unknown." To provide the desirable element of reinforcement, the student is periodically asked to summarize the frames he has read, then to compare his summary with one elicited from the original programmer.

As a simple example, consider the interrogation dialogue below:

<u>Machine Statements</u>	<u>User Responses</u>
What is the name of the Input File?	HOURLY TEMPERATURE READINGS
How many records are in the file?	24

* Perhaps ideally, he would do this by light-gunning copy as it rolls by on a CRT. We use typewriter I/O in which each item of report copy, contributed by the narrator as a response, is numbered. The editor flags the response by typing the number. Report copy not contributed by a narrator cannot be changed except by an executive programmer.

Machine Statements

User Responses

Justify your estimate.

READINGS ARE
TAKEN ONCE PER
HOUR, THROUGH-
OUT THE DAY

A tutorial version of this information can be assembled and might be:

The input file is HOURLY TEMPERATURE READINGS.

It consists of 24 records, estimated as follows:

READINGS ARE TAKEN ONCE PER HOUR, THROUGHOUT THE DAY.

Clearly, this frame results from inserting specifically requested responses into a skeleton frame which reads:

The input file is _____. It consists of _____ records, estimated as follows: _____.

The programming logic of the instruction process is almost identical to that of acquisition. Again, an executive program is provided which states the conditions under which the individual frames are to be displayed to the user. The same executive program is not used for both, however, since the sequence of presentation of the stored data may be different from the sequence in which it was acquired. The executive program may give the user an opportunity to direct the presentation of the course material through his responses to branch-directing frames. This feature permits browsing on the part of the user to the degree permitted by the executive programmer when he developed the frames and the selection logic.

Utility Mode. The utility mode of CAINT encompasses all of the normal file processing activities of a conventional information retrieval system, providing a system user access to the data base to retrieve individual items or arrays, and to make modifications to the files. Although this mode represents a large share of the total programming effort for CAINT, it will not be discussed in detail here since the techniques employed are conventional.

Executive Programming

Executive programs must be written prior to any use of CAINT in conversational mode. These, while changeable, need not change often. In a management

reporting system or photointelligence reporting system, for example, an executive program could be considered almost permanent. There are two facets of executive programming—writing skeleton frames and specifying the conditions under which a frame is to be used. As in man-to-man interrogation, the skill of the interrogator in deciding what question to ask, when to ask it, and how to frame it is paramount to the success of the process.

The executive program provides both an opportunity and an obligation for someone to exercise policy, or broad procedural, control over the CAINT process. While such control can be exercised in any program system, executive programming enables this control to be wielded without recourse to computer programming or re-programming. The language of the executive program resembles a subset of a high order computer programming language, such as FORTRAN or PL/I, but has many fewer operations and is easily learned by the nonprogrammer. More importantly, the executive program is easily modified in its own simple language.

The executive program makes use of a set of skeletonized messages, or frames, which will become its vehicle of communication with the CAINT system user. Writing these skeletons is an important part of executive programming, and is the portion that most resembles instructional programming. In effect, the executive programmer writes a generalized, parameterized program of frames. He then writes an executable program, quite similar to a computer program, to specify how and when to use the skeleton frames.

When the executive programmer knows that, under some stated condition, he will always want to “say” the same thing, he need not skeletonize his program. Thus, a frame always used at the beginning of an interrogation might be “Enter your name,” and this is complete in itself. On the other hand, a report to a narrator of data base changes might be worded as follows:

Item _____ has changed from _____ to _____

or

Items _____ have changed from _____ to _____

These illustrate two kinds of variation—inserts to the frame to be made from the data base, and changes in the wording of the frame as a function of data base items, in this case the number of items changed. We may rewrite these frames as a single, skeletonized frame:

1. Item/Items
2. *
3. has/have
4. changed from
5. *
6. to
7. *
8. .

These segments constitute a skeleton frame of eight subframes. The first subframe illustrates a wording variation. The executive programmer specifies that one of the choices offered here be selected, depending on whether the number of items changed is 1 or greater than 1. The subframes consisting of an * illustrate completion options. Each * is to be replaced with the appropriate information retrieved from the data base, in this case the names of items changed, and their original and final values. Subframes 4, 6, and 8 have no associated options. They are always used if the frame is selected.

The operable portion of an executive program performs two functions. It decides, within a major, subject-related group of frames, which specific frames to use in any specific instance. This is called *selection*. Then, it exercises options on word use or insertions into subframes. This is called *completion*.

It consists, basically, of a set of conditional statements, followed by action statements of the general form:

```
IF (condition on data base items)
  THEN (call frame, increment counter)
  ELSE (go to next condition statement)
```

Conditions are stated in terms quite similar to PL/I or FORTRAN IF statements; as IF ($A > B \cup C = 0$) where A, B, and C represent data base items. The action statements call for such operations as selecting and completing a frame, incrementing a counter, or branching to another executive program statement.

To illustrate, we first suppose that we are in the report compilation activity, and are working on the part of the report where data base transactions are

covered. Within the set of frames that are concerned with these transactions is the one we have illustrated above. It will be selected for use only if the data base item NO_CHANGES (TIME_PERIOD)* is greater than 0. Completion uses a different attribute of the field NO_CHANGE (TIME_PERIOD), its value relative to 1, to make the wording choices in subframes 1 and 3. Also used are the list of items changed, the list of corresponding original values, and the list of new values.

CAINT COMPUTER PROGRAM LOGIC

Program Organization

The system to be described has been implemented as a working, laboratory model. This is a model of a longer-range design planned for a System/360 time-sharing computer. The preliminary system was built around existing programs, primarily the Computer Assisted Instruction Operating System,² and operates on an IBM 1440 computer, using IBM 1050 consoles for communication. We shall describe the logic of the system in terms of our System/360 plans, rather than the early model.

Several computer programs are involved in the processing of the various CAINT modes and activities. However, the main thread of CAINT processing is performed by one program—the *executive program interpreter* (EPI). In general, EPI retrieves conditional statements contained in the executive program, evaluates them, and takes the action indicated in the appropriate action statement in the executive program. The primary action is that of displaying a frame on a console. Included in this action is the process of frame completion.

The completion process basically consists of making choices among a variety of subframes and filling in blank subframes with information from the data base, including prior responses to frames. The conditional statements for determining which subframes to use and which values to insert are identical in form to those used in the executive program. Therefore, using the same form of conditional and action statements discussed above with the addition of a CHOOSE action statement to indicate a subframe choice and an INSERT action statement to indicate the filling of a blank subframe, the completion lan-

* The number of changes in a given time period. The variable TIME_PERIOD is an index on the variable NO_CHANGES.

guage is the same as the executive programming language.

Let us examine how the various CAIN modes and activities make use of the executive programming interpreter, which is the nucleus of the CAIN programs.

The principal processing of the control mode is accomplished via the EPI. Selection of the system activities to be performed is accomplished by the user at the console, responding to frames which describe the choices open to him. The two major areas of control mode processing that require additional programming are (1) console input/output and (2) mode switching. Console input/output routines are included in the control mode because they might be invoked from any other CAIN program, and it is economically sound to have them reside in high-speed memory at all times. Since the control mode programs are also designed to be resident programs, the two were combined. These routines simply display frames to the console and accept responses when appropriate. Mode switching requires a program to interpret mode switching commands, save data on the interrupted process, initiate the new mode, and, after completion, restore the original process. The normal start where no interrupt is involved is the same process without the store and restore steps and is thus accomplished by the same computer program.

The acquisition or narration activity is performed using the EPI plus a set of file processing routines. These routines perform the function of retrieving and storing information in the data base as called for in the executive program. During the narration activity, additional information, in the form of user responses, is acquired and stored in the data base and, therefore, is available to the remainder of the executive program.

An Example

In order to illustrate the program logic, we consider an application of CAIN to the extraction of information from aerial photography by military intelligence photo interpreters. Usually an interpreter has no computer-based system files from which to retrieve known information about the area under surveillance, nor has he any specific directive concerning the precise intelligence information requirements. Of course, he knows a great deal about what types of

objects are likely to be militarily significant and what are not. However, beyond this initial observation, the information derived from the photograph will be quite heavily dependent on the particular interpreter and his point of view. In short, today's problems of photo interpretation for military intelligence may be categorized as follows:

1. *Entry of redundant information.* There is no effective feedback from the information user to the collector; hence, the same information is often collected repeatedly. This has obvious implications on the amount of unnecessary message traffic sent and also on the amount of time wasted by the photo interpreters. The latter is quite important since photo interpreters are highly trained people, always in short supply.

2. *Omission of needed information.* Again, without some form of two-way conversation, the photo interpreter may neglect to report elements of information derived from the aerial photography which, on any given day, may seem unimportant to him, but are critical items to someone else.

3. *Lack of consistency.* The somewhat loose control of the activity brought on by the one-way communication leads to inconsistency in the choice of items reported by various interpreters, and in the language and format in which the items are reported.

4. Report preparation is currently one of the most time-consuming tasks of the interpreter. Reports are compiled and written manually and must be formatted to the specifications of the recipient. In most photo interpretation activities there are individuals whose responsibility it is to ensure that the reports issued by the interpreters are accurate in content and format. These individuals act in this capacity as editors.

5. In military organizations, particularly, there is a large turnover of personnel. New people must be instructed on current strategic or battlefield situations and indoctrinated in the local operating procedures of the organization. In the field, where the accent is on mobility, the terrain and battlefield situation being photographed change often, thus, even the veteran interpreters may need instruction.

The foregoing is, of course, quite generalized but is stated to give a framework within which CAIN could be applied and within which the logic of the CAIN programs can be described.

A sample of typical messages that might be displayed during the acquisition activity in the photo interpretation application follows:

<u>Machine Statements</u>	<u>User Responses</u>
1. Are there any tanks visible on the photo?	YES
2. How many?	1
3. What kind?	M-48
<hr/>	
4. Are there any mortars?	YES
5. Give coordinates for each type, heavy, medium, light.	Q FIND CHARACTERISTICS HEAVY MORTAR
6. Consists of: tube 4.2 inch diameter, 5 feet long, base plate, tripod.	
7. Give coordinates for each type: heavy, medium, light.	HEAVY: UTM FR741324 LIGHT: UTM FR742324
<hr/>	
8. POW reports: CP BUNKER ON HILL AT UTM FR 744320. Can you confirm?	NO
9. Are there any fortifications visible on the photo?	BUNKER
10. Give coordinates of: BUNKER	UTM FR741325
11. Indicate condition by appropriate letter(s)	
a. destroyed	
.	
.	
.	

This particular sample represents just a small portion taken from the middle of a much longer acquisition. Frames 1, 2, and 3 might have been selected because of a stored message from higher headquarters stressing the importance of information about tanks in this area. Note that the interpreter was not asked for a precise location of the tank. This is because the executive programmer has decided that the location question need not be asked for what are normally mobile items. Frame 4 is displayed because of a standing intelligence requirement to report locations of all mortars. Frame 5 requests location information for mortars by type. The response to frame 5 is a query (the Q is assumed to switch the system into utility mode) for the characteristics of one of these types, the heavy mortar, that will assist the interpreter to identify a particular mortar he sees. The retrieved information appears in frame 6. Following this, the original question of frame 5 is repeated, as frame 7, and it elicits two answers.

Frame 8 is an example of how data base information from other sources may become part of a CAINT frame. The statement "POW REPORTS CP BUNKER ON HILL AT UTM FR 744320"* has been inserted into the frame, although it was actually collected during a prisoner-of-war interrogation at some previous time. Frame 10 shows the use of a prior response in a frame. The word BUNKER was the response to frame 9 and was then inserted into frame 10.

The executive program for the selection logic for frames 8 through 11 is shown below.

<u>Statement No.</u>	<u>Statement Text</u>
9	IF SCALE > THRESHOLD & AREA_FOR-TIFICATIONS = 1

* UTM stands for Universal Transverse Mercator which is a grid coordinate system used by the Army for maps. FR 744320 is the actual coordinate, in which 744320 is a displacement (x coordinate = 744) from a point in the earth's surface designated by FR.

Statement No.	Statement Text
	THEN CALL FRAME 8; ELSE DO: IF SCALE > THRESHOLD THEN DO: CALL FRAME 9; GO TO 11; END; ELSE GO TO 12; END;
10	IF RESPONSE 8 = 'NO' THEN CALL FRAME 9; ELSE DO: CALL FRAME 10; CALL FRAME 11; GO TO 12; END;
11	IF RESPONSE 9 ≠ 'NO' THEN DO: CALL FRAME 10; CALL FRAME 11; END; ELSE ...
12	IF ...

The scale of the photo is compared with a pre-determined threshold, representing the minimum scale at which certain objects may be identified, and a check is made for a prior indication of fortifications in the area. In the example, both conditions are met and frame 8 is displayed to ask for confirmation of previously acquired information. The CALL FRAME 8 statement also causes the completion of the skeleton frame. In this case, the text "CP BUNKER ON HILL AT UTM FR 744320" and an indicator specifying the source as a prisoner of war report are retrieved from the data base in order to complete the skeleton frame. If there had been no prior indication of fortification in the area, but the scale had been over the threshold, frame 9 would have been displayed to ask a more general question about the presence of fortifications on the photo.

Since the responder could not confirm the POW report, frame 9 was displayed next to see what fortifications are present. A YES response to frame 8 would have caused immediate selection of frames 10 and 11 to acquire more specific location and condition information.

Since the response to frame 9 was not NO, indicating a positive response, frames 10 and 11 were displayed to follow up. Otherwise, no frames would be selected.

System Outputs

There are two major outputs, other than those retrievable in the utility mode. These are instructional materials and printed reports. Instruction is almost

identical to acquisition from the programming point of view. The same type executive program is used and the same interpreter program. Of course, in the instruction process, what were originally question frames are reworded in declarative form, and are combined with the responses through the frame completion logic. These instruction frames are displayed one at a time to the user at the console.

Report preparation is similar to the instruction activity except that the report is printed off-line and is intended more as formal documentation of the data base rather than as instructional material. The same EPI program with a different executive program is used for reporting, for a report is a sequence of frames printed without user interruption.

The following is a possible report generated from the information obtained during the interrogation, plus previously collected information and specially prepared titles:

PI Report—Print 109VV 67°51'N 67°32'E
4 July 1965

Line

1. Previous prisoner of war report (Reference
2. PW 710) of command post bunker on hill at
3. FR 744320 could not be confirmed.
4. A camouflaged bunker was identified as
5. under construction at FR 741325.
6. ** Weapons Reported **
7. 1 tank M-48
8. 1 mortar Heavy FR741324
9. 1 mortar Light FR742324

The sample report is divided into a narrative and a tabular section to indicate that either, or a mixture of both, is a possible output. It should be noted that, in addition to appearing in the printed report, all of this information resides in the data base for subsequent queries, reports, frame selection and completion, etc.

Editing has been previously described as a combination of several other activities. The one additional program needed is that to translate edit commands into acquisition parameters, if required, and to do simple text editing.

The following editing example shows how two out of many possible editing situations would be handled.

Machine Statements	User Responses
1. Give change command.	INSERT AFTER "CONFIRMED."

<u>Machine Statements</u>	<u>User Responses</u>
2. Enter change.	CLOUD COVER PREVENTED OBSERVATION.
3. Give change command.	CHANGE LINE 5 "FR 741325."
4. Give coordinates of bunker.	UTM FR 741330.

The first two lines represent the commands to the system to insert the sentence "CLOUD COVER PREVENTED OBSERVATION" following the word "CONFIRMED" in line 3 of the report. This is only an insertion of additional information and represents no change to the data base or any CAIN activity. Line 3, however, requests a change to correct some erroneous data acquired during the acquisition activity. In order to effect this change, CAIN reenters the acquisition activity to again display the frame requesting the information (line 4). The corrected response is then given. The report would appear as follows after editing:

PI Report—Print 109VV 67°51'N 67°32' E
4 July 1965

1. Previous prisoner of war report (Reference
2. PW 710) of command post bunker on hill at
3. FR 744320 could not be confirmed. Cloud
4. cover prevented observation.
5. A camouflaged bunker was identified as
6. under construction at FR 741330.
7. ** Weapons Reported **
8. 1 tank M-48
9. 1 mortar Heavy FR 741324
10. 1 mortar Light FR 742324

STRATEGY FOR USE OF CAIN

Acquisition

The strategy to use in eliciting information from a responder is very much a function of the individual application. The nature of the questions to ask, the amount of time needed or allowed for an interrogation, the amount of information needed by a responder, and the language restrictions to be placed on responses do not fall into clearly defined categories. In our early experimental work we have discovered a few principles which appear to have broad application, but it would not be at all sur-

prising to find, a year hence, that interrogation techniques are vastly different from those now in use.

One of the first difficulties we encountered was excessively slow interrogation, caused both by slow terminal speed and the wording of our messages. In particular, we found that the frame writers had a tendency to instruct the responder too often in how to reply, for example, asking a question that obviously calls for a yes-no response and taking two lines of print to tell him so. In the course of an hour's work on the console this can be very irritating. We find, then, that frames should be as terse as possible, with reliance on prior education and callable explanations to handle the ambiguous situations that will inevitably arise.

Reliance on a user's prior experience with CAIN is most important. If the frame writer feels compelled to explain himself and offer hints on responding in every frame, the effect on the experienced responder is deadening. For this reason, we have extensive indoctrination courses for use in our program documentation application, and recommend a similar approach to all other CAIN users. Once it is assumed that the responder is already familiar with the frame author's meaning, there can be a great reduction in the verbiage needed in a frame, even for a relatively complex subject, for all we need do is to get across to the responder which question we want him to answer. He can have seen beforehand the explanation of it. When this approach is not feasible, as it will not be in all possible CAIN applications, instruction can be embedded with interrogation, but the result is a slower overall process. The most important thing seems to be for the system user to understand what CAIN is driving at, even when the machine-generated messages are vague, and to know what he has to do to provide the needed information. This skill will come from practice, particularly in seeing how individual responses are woven into a report format.

Another time-saving technique is the concept of "priming" the responder. This term describes the technique used in the following situation. Suppose we wish to ask a fairly difficult or complex question and we know what information the responder will need to answer it, but suspect he will not have these facts at hand. If some of this information is in the system data base, we can lead up to the question with a series of informational frames that provide the responder the information he is going to need, then ask the question. He may need still more infor-

mation and want to switch into the utility mode, but the intent is to minimize the need for this, while still not boring him with well-known or irrelevant data. Priming, then, is a skill that must be developed by the interrogation course author. It is essentially the same skill as is used by the author of a text book, who must plan the sequence in which he introduces material so that there is a logical flow and the student always has on hand the information needed to tackle the new topics.

The interrogee, as does any author, will want to review his text periodically, as he writes it. The CAIN T responder can always scan his previous responses, but, as we have pointed out, these are not always in the best context for review. He may, then, request periodic operation of the report generating activity to provide a review of his most recent few responses in their ultimate context. We shall have more to say on this review process in the next section. At this point we introduce the notion that review, in addition to its value after the interrogation, is of value during the interrogation, as an aid to answering questions, as well as to compiling well-written reports.

Finally, it has been our experience that we must rely heavily upon the user of CAIN T to become skillful in this means of reporting. Our indoctrination programs replace manuals, they do not automatically impart skill.

Report Preparation and Editing

A CAIN T compiled report can be published directly or used in an edit process. The report editor informs the system of the number of the report section or line he wishes to change. When he does so, the interrogation frame that elicited the response found there is retrieved and asked again. A new response is made and it replaces the old response, as stored in the system files. It may be necessary or desirable to repeat more than one interrogation frame in this situation. The priming frames originally used, and possibly some additional ones that will review a topic in interrogation context, may be necessary. It would be very easy, for example, for an editor to forget some of the background information against which a frame was originally asked, when he decides to change his response. This editing technique can be called into use after the entire interrogation is complete, or it can be done partially, at intervals, in the interrogation process, or both.

We have said that a compiled report consists of text taken from three sources: previous interrogation responses, data base items, and skeleton frames. A responder can change only one of these, the responses. However, especially in early development stages, the automatic assembly process might fail to be sufficiently expressive, or may be misleading. Even worse, a response may appear in, or affect, more than one report entry. A desired change in the response to suit one occurrence may not suit the other. To handle these situations, it will be possible to use an escape technique which would allow the editor to change any portion of a report, even non-response material, but the changes would not be stored in the system files. Obviously, use of this mode of operation should be carefully controlled, for it can easily become the easy way out of some difficult writing problems. It would appear, however, to be a necessity.

Instruction and Indoctrination

Indoctrination is the use of true computer-aided, programmed instruction to teach the use of CAIN T. The courses, which may be difficult to write initially, will be relatively stable, changing as patterns of student usage develop or the system itself is changed. As we have mentioned we find that heavy emphasis on indoctrination pays off in faster interrogation, which is more of a benefit for its ability to hold a responder's interest than for mere clock time saved. Hence, all special usages in the interrogation should be explained here, and sample interrogations run so that the responder becomes thoroughly familiar with the basic frames, their meanings, and their contribution to the assembled report, before he starts making operational use of the system.

With this approach, the usual, cumbersome operating manual is unnecessary. A neophyte need only make his presence at a console known to the system and, from there on, the indoctrination program guides him and tests him, until he shows sufficient mastery to go out on his own.

There is always the possibility that a responder, during a regular interrogation, will encounter a frame he cannot understand. When this occurs, we recognize a special response, such as "HELP" which retrieves a longer version of the frame, one written with this situation in mind, having an elaborate explanation of the meaning and purpose of the questioned frame. Having such back-up enables frame

writers to use the shortened frames that keep interrogation moving at a fast pace. It does mean, though, that many frames must be written twice.

In instruction, elicited responses and data base items are again embedded in new skeleton frames which are then used to teach the content of the system files. The sequence of presentation is again controlled by an executive program, following some logical chain inherent in the subject matter. If branch points are encountered in this structure, the learner can be given control over the choice of path, or it can be forced upon him. Whether or not to delegate control can be at the discretion of the executive program writer. We have had little direct experience with instruction to date. It offers a great potential—the ability to create training programs directly from reports or data files. The pitfalls are many, however. We cannot automatically test the learner on his comprehension and use the test results to choose a path through the instructional material—the technique that allows a fast learner to move ahead quickly by skipping some material and a slow one to get the extra remedial and review work he needs—the technique that makes CAI the valuable tool it is. We do require the student to summarize material as he goes, and to compare his summary with one written by the original programmer. The student then must judge whether he has correctly perceived the material.

Still, we can offer the learner control over speed and sequence, the ability to review, and the ability to query. These capabilities, while less than those of a full CAI system, go a long way toward that goal; certainly much further than a conventional technical report. Especially valuable is that training courses are automatically updated almost instantaneously upon a change in the object system. These, coupled with the imagination of the executive programmer, offer a possibility that limited training

goals can be met at very low cost and preparation time.

Conclusion

The essential point of this paper is that man-machine communication can be made intelligent and purposeful, in both directions. In CAINTE, machine utterances are not mere items retrieved in response to a carefully worded query, nor are they diagnostics, however clever. The machine's statements are based upon a rational analysis, by the machine, of what information it needs, under a condition which it is programmed to recognize. The result is a true dialogue—a conversation that enhances the knowledge of both parties.

REFERENCES

1. J. C. R. Licklider, "Man-Computer Symbiosis," *IRE Transactions on Human Factors in Electronics*, vol. HFE-1, no. 1 (Mar. 1960).
2. "IBM 1401, 1440, or 1460 Operation System, Computer Assisted Instruction," Form C24-3253, IBM Corp., White Plains, N.Y. (1965).
3. John A. Swets and Wallace F. Feurzeig, "Computer-Aided Instruction," *Science*, vol. 150, no. 3696, pp. 572-76 (Oct. 1965).
4. M. V. Matthews and Joan E. Miller, "Computer Editing, Typesetting, and Image Generation," *Proceedings, Fall Joint Computer Conference*, vol. 27, pt. I, Spartan Books, Washington, D.C., 1965, pp. 389-98.
5. Michael P. Barnett and K. L. Kelley, "Computer Editing of Verbal Texts, Part 1. The ES1 System," *American Documentation*, vol. 14, no. 2, pp. 99-108 (Apr. 1963).
6. Jerome P. Lysaught and Clarence M. Williams, *Programmed Instruction*, Wiley & Sons, New York, 1963, p. 2.

SOME PROBLEMS IN DATA COMMUNICATIONS BETWEEN THE USER AND THE COMPUTER

L. A. Hittel

General Electric Company, Phoenix, Arizona

INTRODUCTION

The ever increasing utilization of computer complexes from remote sources imposes a new dimension to the almost overburdening task of systems design. Remotely accessed systems require a tight integration of communications services and equipment in order to effect a system which is both efficient and responsive.

This paper is a tutorial attempt to acquaint the computer system planner with the extra segments of system implementation necessitated by the desire to serve remote users. Time-sharing systems, direct-access systems, and management information systems will require an ever increasing penetration into the world of digital communications.

Four chronological phases in the development of a communications-oriented computer system will be examined.

The first phase is the establishment of a communications plan. The idea of the communications plan is to document the choice of functions, the choice of facilities, the planned capacity, the necessary test features and terminal equipment required to implement the system.

The next phase covers the period between the ordering of the system and the installation of the system. This time period should be utilized to develop the necessary programming required for operation and control of the communications equipment.

The third phase devotes attention to the installation itself. Pre-installation preparation and common installation problems are discussed.

Phase four has been chosen as the operational period. In this section of the paper attention is given to operation and maintenance of a remotely accessible system.

This paper is designed to provoke thought among those who build systems in a "batch" environment. It is dedicated to those rugged individuals who now wish to tackle on-line, direct-access, or information systems.

THE COMMUNICATIONS PLAN

Like any system plan, it is not complete unless all parts of the system have been fully considered in relation to each other. Many designers treat the data inputs and outputs as peripheral functions and assume that they have little control over the choice or the operation of this piece. This is hardly the case. Nowhere else in the system is there such a wide selection of methods, services and equipment.

Improper use of communications can result in excessive cost and poor service to the system users. The following factors must be evaluated in order to arrive at the system communications needs:

1. The average time that each terminal-to-computer connection will be in use. This is called "holding time."

2. The urgency of each computer access request. How long can you wait for the computer service?
3. The amount of output that must be printed and how fast it must be completed. The printing speed will often govern the choice of a given data rate.
4. The number of graphics (characters) which must be input and printed on the terminal. Large graphic sets along with the required control code characters will need a corresponding number of bits to describe each character uniquely. ASCII has provisions for 94 printing characters.

I have tried to organize a chart of data services useful to the time sharing designer. For simplicity of explanation I like to separate data communications into three speed classes:

Class One—Up to 30 characters per second. This class is used for keyboard-printer terminals. Nominal speeds are 6 characters per second for older communication terminals, 10 characters per second for newer terminals and 15 characters per second for typewriters and proposed communication terminals. The typical data set limit is 30 characters per second.

Class Two—100 to 300 characters per second. This is the normal telephone circuit. The speed differences are a function of the quality of the circuit and the sophistication of the Data Set. The better circuits and the more sophisticated Data Sets are priced accordingly.

Class Three—1000 to 10,000 characters per second. This class is still in the early development phases. American Telephone and Telegraph Company offers a wide-band data service using their TELPAK circuits with special Data Sets. Western Union has a limited availability of wide-band circuits and experimental Data Sets. The prime use of this class of service is for high speed terminals such as line printers and magnetic tape stations. Computer to computer transmission will also use this type of service as the need for the interconnection develops.

The aforementioned circuit speeds are available in variations of switched (dial) service and private line service from common carriers, such as AT&T, Western Union, and General Telephone. Switched network services which are useful for time sharing connections are:

TELEX. This is a combined domestic and international dial service offered by Western Union. This service is useful for connecting teleprinter terminals to computers. The service is limited to 50 words per minute. The available graphic set is limited.

TWX. This is a domestic dial service offered by the American Telephone and Telegraph Co. It is useful for dial service between teleprinters and computers. Two speeds are available: 60 words per minute with limited graphic set and 100 words per minute with an improved graphic set. Availability of this service is excellent.

Dataphone. This is a domestic service offered by AT&T through its operating companies. Data Sets are provided to allow an alternate use of the existing public telephone system. It is useful for dial service between various terminals and computers. The unmetered use of metropolitan service is very convenient for local terminals. WATS can be used for high-usage long distance service. Speeds vary as a function of the Data Set used. 30 characters per second, 120 cps. and 300 cps. represent typical limits.

Foreign Exchange Service. This is a telephone service offering which allows a computer to have telephone numbers which belong to a locality other than the one in which the computer is located. This allows inter-city or inter-zone operation on a fixed cost basis.

TELPAC. This is a bundle rental plan for Bell System circuits. The bundle sizes are 12, 24, 60 and 240 voice grade circuits. It is possible to use all or part of the bundle with a suitable Data Set for high speed data transmission. TELPAK can be used in conjunction with Foreign Exchange Service or for interconnection of two Private Business Exchanges.

Broadband. This is a Western Union dial system for data transmission. The data rates are variable and selectable. In several cities bandwidths of 2kc and 4kc are available. Bandwidths of 8kc, 16kc and 48kc can be obtained in a few major cities. Broadband service is useful for short-to-medium duration transmissions between high-speed terminals and computer centers.

DATEL. This is a new international data transmission service being offered by ITT World Communications Inc. DATEL 100 is a full-duplex 100 bits per second facility. DATEL 600 can be used for 600 to 1200 bits per second transmissions. Additional DATEL services are being made available.

The problem of choosing the best service for your applications may seem insurmountable after scanning the above list. I will try to give a few guide lines as an insight to the problem.

If you have most of the terminal devices located within a "private" geographical center, such as a building or a complex of buildings, consideration should be given to installing the entire system without using common carrier services. This is a "pioneering" attitude but has economical advantages.

There can be specific disadvantages to this method which should be considered. Unfortunately, not many computer manufacturers can supply and maintain the required hardware. The installation costs of proprietary computer links can vary extensively as a function of the building type and local building codes.

The major cost savings are in the elimination of separate Data Sets or "modems." On short "loops" or circuits within a building, the extra sophistication of a Data Set designed for long distance is not needed. Some replacement is required; but it usually can be integrated into the terminal electronics on one end and the computer channel on the other end.

A variation on this method is to use common carrier supplied "loops" with integral line couplers. This is useful where installation costs are prohibitive. It also allows the circuits to be extended beyond the boundary of the private property.

Customer-supplied Data Sets can be used on all private line common-carrier circuits, provided they meet the common carrier interface standards.

An interesting combination of the previously mentioned techniques can be used where a large number of terminals are to be located in each of several buildings. The data terminals can be connected to a concentrating device with customer-owned wires and couplers. The concentrating equipment can be connected to the computing facility using common-carrier supplied circuits and Data Sets.

When only a few terminals are located in a concentrated area, but many terminals are to be connected to the computer facility, an entirely different approach is needed. The use of a switched-network

service should be investigated as a function of cost. In order to calculate cost, more information on the use of the terminals is required.

"Holding Time" is a term used by the communication people to indicate how long a circuit is in use. If the holding time is in the order of seconds, a dial circuit may be too time-consuming to use. More time would be devoted to completing the computer connection than devoted to the computer operation.

If the holding time is a few minutes for each terminal to computer operation and only a few connections per day are needed, the lowest cost dial service should be sought. If the holding time is long for each terminal connection, but only a small number of connections per month are required, a dial service should still be investigated. Inward WATS at the computer could be used to reduce toll costs. If the geographical distribution of the terminals is such that time charges would not be in effect if Dataphone were used, the long holding times would only require more computer access lines. If the time charges appear to be excessive, then an evaluation should be made of a private line circuit from a common carrier versus a switched network circuit. WATS can reduce toll costs.

WATS (Wide Area Toll Service), Foreign Exchange service, and CCSA (Common Carrier Special Applications) networks will be less costly if properly used with dial service. When many circuits are required between the foreign point and the computer, TELPAK can be used with Foreign Exchange service to hold down the cost.

I mentioned concentrating devices before without explanation. These devices may range in complexity from simple wire-multiplexing equipment to stored-programmed communication computers. In the wire-multiplexing types, both frequency-division and time-division are used. Frequency-division devices take a wide-band circuit and break it into several smaller band channels. Some bandwidth loss results from the unused space needed to separate each channel. These lost segments are referred to as "Guard Bands."

Time-division devices interlace data bits from each of several lower bit-rate channels on one higher-speed channel. In order to accomplish this simply, the interlacing must be done synchronously. The high-speed bit-rate must be greater than the low-speed bit-rate times the number of low speed channels. A loss on the time-division equipment results from the need to add reference and synchro-

nizing bits. When terminals having mixed bit-rates need to be concentrated, synchronous or time-division concentrators cannot combine the varied bit-rates by definition. In order to concentrate mixed bit-rates on a frequency-division concentrator, each sub-channel must be designed for the bandwidth necessary to transmit the required bit-rate. Another alternative is to select a single sub-channel bandwidth which will pass the highest required bit-rate and use it for all sub-channels. Communication computers are useful when the circuit costs are high in relation to the cost of the computer. A stored program device can usually provide a much higher concentration ratio by using short-term buffering. Since most terminals are not used at maximum bit rate continuously, it is possible to average the combined data rates. This is accomplished by operating the computer in a fast store and forward mode.

PROGRAMMING FOR DATA SET CONTROL

One hardly equates programming and Data Set operation until he has attempted to operate a system that is remotely accessed through a switched network. Most Data Sets have automatic answer and automatic disconnect features. These automatic features operate well provided that human supervision and intervention is available to handle exception cases.

When a large number of Data Sets are connected to a computer, human supervision is neither practical nor possible. Figure 1 lists some conditions that would require supervision above and beyond the automatic features found in the Bell System's 103 Series Data Sets. Class 1 conditions can be detected through monitoring of the distant carrier at the appropriate times. Class 2 conditions can be detected through an elapsed time check from the last transmission received.

Class One Non-Spacing Disconnects

1. Circuit loss due to network failure.
2. Abrupt terminal disconnect (EOT).
3. Power failure at remote end.
4. Short disconnect (less than 1.6 sec.).

Class Two Loss of activity on the channel

1. Broken tape on sender.
2. Failure to disconnect at end of job.
3. Terminal operator called away.
4. Electronics failure.

Figure 1

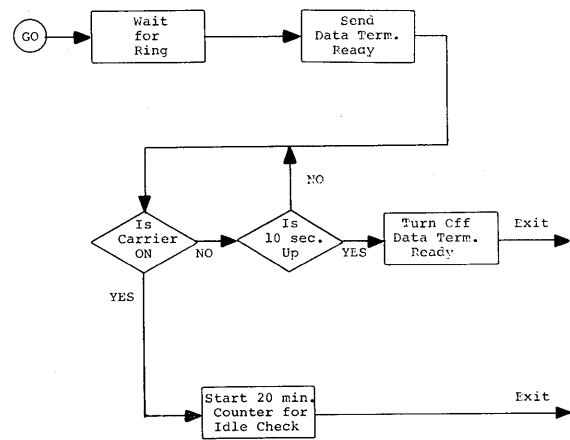


Figure 2

Figure 2 illustrates a typical block diagram for the necessary logic to provide Class 1 protection using the 103A Data Set. When starting from an idle condition, the channel is checked for a ringing signal every second. If the ringing signal is detected for a channel, the "Data Terminal Ready" signal is sent to the Data Set. This signal causes the Data Set to answer the phone. After a sufficient time is allowed for completion of a "handshake" between Data Sets (about 10 seconds), the Data Set is checked for "Carrier On." This indicates that a data carrier is being received from the distant station. If the carrier is not on, one can assume that the calling party does not have a compatible Data Set (if any) or has "hung up." In this latter case, the Data Terminal Ready signal is turned off and the Data Set will go "on hook."

When carrier is detected upon completion of the answering signal, an elapsed-time counter is initialized. This counter is used to shut down the circuit if no data is received within an established limit of time such as, say 10 to 20 minutes.

Figure 3 shows a protection scheme for circuit loss. Every one second the channel is checked for incoming carrier being present. When a carrier loss is detected, a seven-second time-out is started. If the carrier returns within the allotted time, the counter is reset and the operation resumes. If the carrier is still off after seven seconds, the Data Terminal Ready signal is removed, causing the Data Set to "hang-up." When this condition arises, it may be desirable to inform the system executive to save any temporary data for the user until he can reconnect with the system.

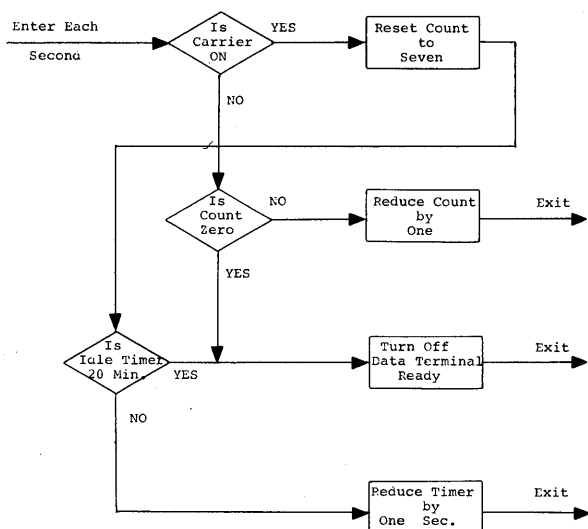


Figure 3

Figure 4 gives an abbreviated list of the channel states implied by the flow diagrams in Figures 2 and 3. When programming for other Data Sets, especially those used in a half-duplex mode (one-way-at-a-time transmission), the carrier check cannot always be used as a guide. Slow-speed Reverse Channels are sometimes used for assurance that the circuit is still connected. If reverse channels are not available, then a time check is needed. This technique requires that each message be acknowledged by the receiving end. Dummy messages are sent to test idle circuits. When an acknowledgement is not received within the specified time period, a circuit loss is assumed.

On private line circuits, circuit supervision is needed for a different reason: identifying the loss of a circuit and reporting that loss to a repair service. It is difficult to discuss communications programming without a word about character sets. There is a strong temptation to restrict the code set within the computer to something less than what exists at the terminal. This is usually done to make the code

State No.	Condition Description	Normal Exit	Branch Exit
0	Idle	Hold 0	1 (Ring)
1	Ringing	2 (answer)	0 (no ans.)
2	Send Ready	3	
3	Check Carrier	4 (carrier on)	3 (time-out)
4	Normal Receive	7 (end job)	5 (carrier loss)
5	No Carrier	4 (carrier on)	7 (time-out)
6	Normal Send	4 (end trans.)	7 (end job)
7	Shut Down	0 (1 sec. delay)	

Figure 4

set map into that which is used for batch data processing (6 bit BCD).

Most of the terminals in use today can produce a printable character set which is equal to or greater than the set used for regular computer input and output. In addition to the printable character set there are many terminal control codes in use today. These codes must be employed in order to effectively utilize the terminal.

The most elementary keyboard printer terminal uses such functions as *idle*, *carriage return* and *line feed*. Other common terminal control characters are:

1. Horizontal and vertical tabulating
2. New form or page
3. Red and black ribbon selection
4. Paper tape reader on-off
5. Paper tape punch on-off
6. Punched card reader on-off
7. Card punch on-off
8. Main printer on-off
9. Auxiliary printer(s) on-off
10. Subscript and superscript line controls
11. Ring bell

When designing a system with a mixed terminal set, it is desirable to provide the applications programmer with a common code set. In choosing the common code set it is better not to have shift characters. Shift characters will require all system programs to use a more complex input scanning algorithm. The disadvantage to increasing the byte size of all characters is that it will increase buffer sizes and decrease file efficiency.

The communications programmer should provide the specified terminal delays required for mechanical functions such as carriage return and tabulation. This may require him to maintain a line character count for each channel in order to calculate the proper delay. Once a communications plan has been drawn and the terminal types have been picked, the system designer must work with the system programmer to specify the programming implementation required. This is necessary in order to operate the data services effectively and provide a common interface for the application software implementation.

INSTALLATION

In the earlier part of this paper, I indicated that a communications plan was needed for any measure

of success. If inadequate planning is done during the system design phases, installation will be chaotic. Perhaps the best use of a good communications plan is for an implicit specification of the common-carrier supplied pieces of the system and the communications hardware vendor supplied items. A large order, if released to a common carrier on short notice, may result in the operation having to accept temporary service of an inferior nature. It takes time to engineer and install a large number of good communications circuits.

If switched circuits are to be used, the capacity of the switching office may have to be increased, or specialized switching equipment may be needed to meet system requirements. If the terminals are located in areas served by different switching offices, the inter-office capacity may have to be increased. A two-to-five year capacity forecast would be useful to the common carrier for his long-range planning. The customer will then have assurance that he can have the service available as he expands his operation.

Terminal options and Data Set options should be reviewed when the order is placed with the salesman and again during the installation with the installers. Many a customer has been frustrated during installation by not having a list of the compatible options among system software, Data Sets, terminals and computer interfaces. Unfortunately, most option information does not appear in the standard literature of either the Computer System manufacturer or the Common Carrier. Be sure to ask; this has been a major problem in many installations.

Plan ahead for a method of testing the installed communications circuit after installation. Do not depend upon unchecked software to do this job. If necessary, prepare some very simple test programs to exercise the terminals, Data Sets, and computer channels. If the communication system permits, terminal-to-terminal checks may be very useful.

MAINTAINABILITY AND OPERATION CONSIDERATIONS

Once a time-sharing system is installed, the problem of operating it and keeping it operational is important. The system application and user relationship will determine the required operational schedule. My advice to the neophyte operations manager is that to provide a satisfactory on-line service, it must be available when the *user needs* it. This often means six a.m. until midnight (executive

schedule) and weekends. In a public system, the operating hours are usually longer. Off-hours are not usually profitable for time-sharing—but neither are they for the common carriers. The operations staff is dealing with an unseen set of users who are constantly changing. The only awareness that the operator has that a system is working is from the communication circuit indicators. A trained operator soon can relate the number of active communications circuits with the time-of-day. It is this type of acquired intuitive judgment that provides today's operators with assurance of correct hardware performance.

It is time to give consideration to an instrumented approach to system checking. With technology borrowed from the airframe manufacturers, instrumented checkout techniques can be implemented within a time-sharing system. Two types of checking are required: system-checking and channel-checking.

System checking can be accomplished by simulating a customer on one or more channels and comparing results with previous tests. An "all is well" message can be printed periodically on the operator's log. Abnormal results can be indicated immediately.

Channel checking can be implemented at two levels of detail. The choice is governed by hardware available to do the job, its costs, and the incentive for exacting tests.

When common-carrier supplied circuits are used with common-carrier supplied Data Sets, it is necessary to determine only if the channel is working properly (GO or NO-GO). It is still desirable to offer the common carrier repairman detailed information as to the type of failure. This speeds the repair of your service.

If dial circuits are being used, then a separate, compatible Data Set with the appropriate originating interface should be installed for checkout purposes. Each working channel is "dialed" and data is passed in both directions and compared for errors. This process can be included in the system software as a fill-in task during periods of low system use. A faulty channel on a straight rotary-selected dial group can guarantee the most trouble for the most users. When the system activity level results in the selection of the faulty channel, errors cause the short-duration use of this channel. As soon as termination occurs, that channel is then made available for the next selection. When this pattern is repeated

several times by the same user he will report the trouble.

When random selection is used, a fault becomes transitory to a specific user and, therefore, is almost never reported. If it is reported, the bad channel is difficult to isolate. If the error is not reported, random trouble is experienced for many days. When error situations are detected on rotary dial channels, the faulty channel should be reported and temporarily removed from use. Patch panels can be used to connect spare hardware to the dial circuit for this purpose.

An alternative to patch panels is to use a "busy-out" feature which can be provided by most common carriers. This feature will prevent the associated circuit from being selected by the switching equipment. Newer Data Sets may allow control of the "busy-out" feature through the business machine interface.

Private-line operation requires specialized hardware to accomplish this channel selection for test purposes. If private-line circuits are used, channel errors will affect only one terminal. Therefore, identification of the faulty circuit is easy.

When the Data Sets are owned by the customer, a new requirement is added to the problem of channel testing. The user now needs diagnostic information which will help to effect the repair. If audio carrier Data Sets are used, a programmed (digitally controlled) frequency generator and amplifier could be used to send test patterns to the Data Set in question. A digital counter could be used to check transmission frequencies of the Data Set. Similar equipment can be used to introduce and measure DC loop waveforms and currents. A step-by-step test program can be included in the system software to drive the programmable test equipment. Quantitative evaluation of each channel's performance could be made. Deviations above preset limits would be noted for maintenance purposes. Perhaps some day in the future, common-carrier supplied Data Set systems would include this type of test facility.

SUMMARY

In closing I would like to recap the four phases of time-sharing communications planning which have been covered in this paper. The first phase was the development of a communications plan. In this phase the designer is directed to examine his system

objectives as they relate to communications. Points to ponder in this examination are:

1. The distribution of the terminals.
2. The amount of time each terminal will be in use.
3. The reliability required for each terminal.
4. The code set and message structure.
5. The human requirements for the terminals.

Once the objectives have been qualified and, hopefully, quantified; the designer should acquaint himself with the communication services available. If he has difficulty with developing a suitable understanding of communications networks, there are many private consulting firms willing to assist. When a solution to his objectives has been found in a set of communications hardware and services, he has a *Communications Plan*. In order for the Plan to be effective, it must be fully documented.

Communications software is the orphan of the decade. Most system designers apply insufficient attention to the problems of:

1. Data Set control
2. Circuit and System checking
3. Human aids for the terminal operator
4. Message formats and line control
5. Graphic sets and control codes

I have yet to meet the system designer who felt that he had adequate software for communication control in his system when it went into service.

System installation has often been considered a manufacturer's problem. When one or more computer hardware manufacturers and one or more communication service suppliers are involved, the final responsibility resides with the system manager. He should be prepared to:

1. Specify all options in the hardware.
2. Advance-order all communications services.
3. Develop a communications testing plan.

Once a system is "on the air," attention is focused on how to make the installation a smooth running one. Time-sharing service is provided for a set of jobs which are unseen by the operator. This is a new phenomenon to most computer center managers. Instead of being able to judge system performance by

the job accomplishments, the operators need a system diagnostic tool that is independent of the user's work.

Although the responsibility for hardware maintenance may belong to the computer manufacturer and the communication service supplier, the systems operations staff cannot neglect their customers.

Therefore diagnostic procedures and hardware testing within the software is a necessary part of the successful operation. A cooperative attitude will speed system repairs faster than finger-pointing. Remember the system users are measuring the overall performance and are not sympathetic to intramural disputes.

COMMUNICATIONS NEEDS OF THE USER FOR MANAGEMENT INFORMATION SYSTEMS

D. J. Dantine

Clark Equipment Company, Buchanan, Michigan

The need for data communications is not restricted to only the very large and very widely dispersed business or service organizations. Small businesses tend to have all the problems of their larger counterparts, especially the problem of the conveying of key financial and operating data to the personnel involved in the daily activities of that business. The extent of geographical dispersion of the business is also not a delimiter of the communications need. The widely dispersed organizations usually will reap greater benefits because data communications will help overcome their problems of geography. But, have you ever considered that a management person could have his desk backed right up to a computer and yet have a severe communications void?

As the reader might already suspect, the term "data communications" is something much more than the pure communications services offered by companies like Western Union and the Bell System. In addition, it must permeate the complete data-processing system of an organization and be an essential consideration in its very design. Then, and only then, can it be a true management information system.

DEFINITION OF MIS

A management information system is a data processing system that augments the management of the business at all levels and in all functions by supply-

ing to the managers, their subordinates and staffs, information needed for control of operations, planning and decision-making. It is characterized by:

1. an integrated, on-line retrievable data base
2. a communications capability inbound and outbound (time and place utility of data)
3. response times compatible with people's needs
4. an optimum totality of essential business data stored and processed in a way that it produces useful information to the managers in a form compatible with their needs and objectives.

The first three of the characteristics of the MIS are intimately tied into some form of communications. Is it any wonder that data processing equipment planners see the future computer systems as 75% communications and 25% central processor? Must we also begin to suspect that most of our systems design and programming in the future will be in the communications and data base areas?

PHYSICAL EQUIPMENT AND FACILITIES NEEDS OF THE USER

Although growing at a significant rate today, the users' communications needs in the equipment and

facilities area are not being completely satisfied. The two most serious impediments to continued progress are the lack of satisfactory communications software and the undue interference by government in the data communications field. The whole area suffers from a lack of good fresh thinking by terminal equipment designers, people who continually restrict and inhibit their designs to accommodate pseudo conventions like punched cards and paper tape that should have been discarded at the advent of the business computer, or at best the year after its advent. We shall look at the needs of the user in each of these two areas separately.

REMOTE TERMINALS

Hardware Considerations

It is not the purpose of this paper to deal with each terminal type separately and at length, but instead, to outline for the reader some thoughts that must be entertained by him in equipping his system with the terminals necessary to make it a true information system.

a. *Suitability to Purpose.* It is hoped that in the foreseeable future that the user will have as much flexibility in the selection of a terminal device to fit his need as the consumer has today in selecting a TV set or electric shaver. They both demand only 115v. 60 cycle A.C. current (standards arrived at thru free and open competition); the manufacturer being a choice of the customer not determinable by the brand name of the generator (the processor in the case of the computer). With the terminal being much smaller and less complex than the central computer gear, there exists the distinct possibility that many manufacturers will be able to compete effectively in the field to the benefit of the user. As such, the user should have the alternative of fitting the terminal to his specific need without the need to resort to general purpose devices or pay the severe penalty for a "customized" device.

b. *General Purpose vs. Special Purpose.* A key consideration here is the user as a person. A clerk performing a high volume repetitious job, like a bank teller, is better suited to a special purpose terminal. On the other hand, the general purpose clerk who might be concerned with some data input, some form typing, some letter and secretarial work plus an amount of computational work would be better suited with a general purpose device. This device

could look like a typewriter but indeed act like a keypunch, compute like a calculator and all in all suit the clerk especially from a cost and efficiency standpoint. The same comments can apply to higher speed devices that might serve a department or section of people.

c. *Economic Considerations.* Terminals, like any other piece of data equipment, must yield a return on investment. However, the user must beware of justifying each and every terminal. The justification must be made on the system as a whole. As an example, a recent decision by a large corporation to move toward a data processing utility time-sharing system requires the use of high speed (240—480 cps) terminals to replace some 12 existing computers. The terminals will be installed in all key functional areas of each product division. Justification of these terminals in each area is impossible. The removal of any one of them leaves the system incomplete. However, the total cost of all the high speed terminals almost exactly equals the cost of the input/output peripherals on the present computers. Thus the terminals are justified in total and all areas are served including some areas not now able to justify and have a computer.

d. *Central vs. Remote Logic and Control.* To date most terminal design has been in precisely the wrong direction. It has attempted to build into the terminal all the logic and control necessary for it to stand alone. Even in this regard it has missed the boat because no terminal designed to date has been able to duplicate the control that can and must be exercised by the central processor at input or output time. A terminal in a MIS rarely stands alone but instead must stand as an integral on-line part of that system. In trying to build this logic and control into these terminals the designers have been forced to price their terminals at a price too high for effective usage.

The production recording terminals seen to date are perfect examples of this design error. Their cost is so high that they cannot be used to directly serve the factory worker. Instead they must be pooled into centers thus forcing a secondary communications system between the worker and the center or else cause the worker to physically move to the center with loss of time and efficiency. Even in doing so the recorder still cannot perform all the audits and checks necessary to validate the transaction. Sure, it can check to see that a card was inserted in the reader (probably a redundant operation), that a

badge was also inserted, and that someone entered the right number of characters of variable information (a form of controlled GIGO). When all this is completed, the computer will still hang when it finally senses that the man completed more pieces than were started in the first place. It thus becomes obvious that the "cheap and dirty" terminal such as a telephone sitting right next to the worker could do a much better job plus have the advantage of having the computer tell the worker what he did wrong before it accepted the input.

The on-line high speed modular terminals with communications-accuracy checking capabilities can also be looked at in the same light when contrasting them to the on-line computer acting as a terminal. These communications terminals are to be preferred for the following reasons:

- They cost only about 50% of that of the on-line computer. If the on-line computer is of sufficient size to handle complex editing and output formatting, the cost could be as high as five times that of the terminal.
- The on-line terminal makes use of the processing capability of a larger central CPU with a much lower unit operating cost.
- The on-line terminal forces all data thru the central controlled data base thus ensuring that there exist the complete data base needed for a true MIS as opposed to the "islands of information" systems installed today.

e. *Environment.* Heat, humidity, power and space considerations must be a part of design considerations. Terminals basically should be designed to exist where people exist today.

f. *Subsets and Modems.* The development of the subset and modem has permitted the more universal use of the terminal for it now can be used wherever phone lines exist. However, the subset has also created a cost factor that at times exceeds the cost of the terminal. Quantity and competition (if permitted to be used on Bell facilities according to reasonable specifications) should bring the cost down. The installation of a MIS usually envisions the use of many terminals at some locations. This subset cost problem could be eased materially thru the development of modular subsets that would serve multiple terminals thus eliminating redundant circuitry. The use

of line concentration and multiplexing possibly combined with modulation and demodulation could also materially reduce these costs. It is not unusual to find the cost of subsets and line termination charges to exceed line costs today on lines of several hundred miles or less.

g. *Flexibility and Adaptability.* Terminals should be designed in such a way that they can be adapted to the needs of the user. The teletype has some of these capabilities built into it. During the past year one user had requested that Bell change their teletypes to print the end-of-message character which in the standard machine does not print. The answer was, "It could not be done." Further investigation with a teletype engineer proved it was possible to do it. The problem turned out to be more one of administration than one of engineering. The part involved perhaps costs 15¢. The real problem is one of filing of special tariffs in all 48 states. Here is a case where undue government regulation almost makes the satisfaction of users' need impossible. The purpose for regulation is to protect the consumer in the case of monopoly or utility situations. However, the teletype attached to a subset is not a product just of the monopoly. I can rent it from either Bell or Western Union. If I don't like a teletype I can rent an IBM 2741 on-line typewriter, or a Dura, or you name it. Here lies a real problem that must be resolved realistically and to which the FCC must turn their attention so that their policies do not become a stumbling block to data communications progress.

Software Considerations

Software to date has little supported the terminal area. The cost of programming of communications-based systems has been a more portentous consideration than the cost of the hardware in many instances. A fact that makes this weakness of today's systems so frustrating is that much of the communications software need not be custom designed for each installation. Much of the software could be composed of standard sub-routines that could be combined in a modular fashion along with the user's connecting logic to yield the would-be custom system.

In the main, communications software is now following the same evolutionary paths followed by the general purpose computer assemblers and compilers. Again computer manufacturers have relied on the customer to hack his own path using only crude assemblers working almost at the 1:1 level. Diag-

nostics have been almost non-existent as a part of operating software and very weak as a part of maintenance software. As a result needless time is spent determining the cause of malfunctions; whether they exist in the program, the hardware, the subsets, the facilities or the terminals. It is with these thoughts and experiences in mind that the writer recommends the following software considerations as basic requirements for future computer systems.

a. *Communications Executive for Remote Programmable Terminals.*

1. Provide communications accuracy control.
2. Provide terminal diagnostics with results being output at the control point.
3. Provide bootstrap ability for program overlaying from central CPU.
4. Eliminate the chances for operator error.
5. Provide for both attended and unattended operation.
6. Provide for diagnostics not only of the terminal but also for the modems and facilities between CPU and terminal itself.

b. *Communications Executive for Non-Programmable Terminals.* Be essentially the same as above but built-in as either hard-wired logic or hardware design.

c. *Central Processor Software.* The central processor software should have the following characteristics:

1. It should be a part of the operating system if the communication devices are to operate in on-line real-time mode.
2. It should contain all needed sub-routines for the control over each terminal and each communications facility or device utilized including diagnostics.
3. The communications operating program should, for the most part, be able to be written and compiled in a high level language producing efficient coding and linked to the operating system.
4. It should provide for operation in either the reactive or batch mode at the option of the user with terminal considerations in mind.

5. The operating system used will be assumed to include an adequate data base establishment and maintenance system to support the reactive stations, retrieval devices and on-line real-time data entry devices.

Human Considerations

The human considerations in the design of terminals are quite different from those of the processor itself. With a processor, the people involved can be selected and conditioned to adapt themselves to the unit. The opposite exists with terminals. Here we no longer have a select group but instead the complete cross section of all employees from the factory worker through the clerk to the busy executive. They all have one point in common and that is that they will resist any amount of forcing them to adapt to a terminal. As such, the terminal must fit their environment, their mentalities, and if possible, become a part of their work station. To these ends designers must consider such things as:

- Ease of training.
- Acceptance.
 - Does it make their work harder or easier?
 - Is it a close relative of devices they encounter in their everyday activities?
- Heat, noise, smell.
 - Does it make their working conditions worse and as such impair acceptance?
- Accessibility.
 - Can the person get at the device to solve his problems or because it is non-accessible will he continue to use manual or less efficient methods?

CPU Considerations

The use of terminals has a definite effect on the type of processor that must be utilized. In the main, none of the third generation computers are specifically designed and built for communications. The only exceptions have been recent announcements in the time-sharing area. These computers would better be separated into a class by themselves since their similarity to other models of their line seems to end with the model series number. In spite of not being

designed specifically for communications, the third generation hardware can do a creditable job given adequate software and the use of communications hardware options. Some of the hardware features a user will require and should look for are:

a. *Buffers/Communications Controllers*. In order of effectiveness these fall into three distinct classes. They are:

1. *Hardware Buffers or Interfaces*. These units essentially have the capability of receiving characters or words over a communications line and storing same. Most other work must be accomplished by the processor itself. In a fairly large system this process work alone could occupy 100% of the capacity of the smaller models of even third generation computers.
2. *Communications Controllers/Computers*. These units are computers in their own right except that they might not contain all the computational abilities of GP computers. However, they might have bit handling characteristics not found in small computers. These units handle the entire communications job exclusive of processing and storage of the data, the latter being handled by the GP processor to which they are attached.
3. *Integral Buffer/Controllers*. This unit is an integral part of the input/output controller of the CPU and terminal communications are handled with all of the ease and by using the same interrupt and memory access schemes as used for regular peripherals. In the more sophisticated models of these units the special bit and character manipulation necessary for communications (unlike peripherals) is handled by hard-wired micro logic built right into the I/O controller. As such, the unit uses little more processor time for the handling of communications terminals than it does for peripherals.

All three of the above devices can be effective as long as the first two are adequately supported by software. In the first, the user pays the price of high

CPU time for communications. In the second the user pays the price of additional hardware and some CPU utilization. The last situation is that of hardware almost exclusively. The above devices, in the order listed, will normally be added to small, medium and large scale CPU's.

b. *Memory*. On-line real-time communications increase computer memory requirements unless one wants to pay the price for overlaying or paging. This additional memory is needed for the added size of the operating system that must be employed, the buffer areas that must be provided and the additional communications and processing programs that must be in core at any moment. A trade-off available to the user to offset the large memory requirements is the use of hardware paging techniques as found in the pure time-sharing systems just being announced.

c. *Multi-Programming/Time-Sharing*. As soon as on-line communications capabilities are added to a system the need for having more than one program resident in core at one time becomes mandatory. The larger and more complex the MIS and the organization being served by it, the more programs that must be in core at one time. There are two solutions to this problem. The first is that of multi-programming. However, as soon as more than about 10 programs are in core at one time the control job imposed on the operating system seems to become so severe that software ceases to be an acceptable answer. The addition of a second processor to the system increases the capability of running more programs, but only to a minor extent. With it the software job becomes even more complex. The only logical solution to the problem seems to be the substitution of some hardware functions for the operating system's relief. Thus, with the hardware-oriented time-sharing systems we may expect the number of simultaneously run programs to materially increase plus effectively be able to utilize the capacity of additional processors, memory controllers and I/O controllers.

After reading some of the above, the very small user could have become even more disenchanted than at the outset. This need not be the case. The very small, and even users of single medium scale second generation equipment, might well consider the purchase of time and facilities from a data processing utility. Already we note signs of some of these organizations coming into existence. The West-

ern Union organization appears to be making material strides in this area. Not only will they be able to supply the computation service and terminals, but will be able to supply the facilities as well. We can be sure that many of the present scientific-only time-sharing systems will in the future be supplanted by time-sharing systems that will offer business data processing and storage services as well.

FACILITIES

The second major area is that of facilities or those equipments and/or services provided to the user by the common carriers or by the user himself, should he provide his own lines or microwave system. Some business organizations which are physically located totally at one location may not need to initially concern themselves with facilities and as such only be concerned with terminals. However, at some later date, with the advent of information storage utilities and concepts like on-line banking, all users will eventually have to consider such interconnections.

Considerations of the Line Facilities

Considerations of the line facilities are assumed to include not just the pure rental of the lines, but the connections and special adaptations necessary to permit these lines to operate with data. Items that must be considered are discussed separately in the following paragraphs.

a. *Suitability to Purpose.* There are a great number of different types of services available today. It would appear that the user is more than amply served. However, a look at the numbers and types of terminals already announced, plus those being worked on might indicate that a shortage of facilities exists. The actual condition might be more one of inflexibility of services offered rather than shortage.

The tendency has been to group facility offerings so as to better comply with tariff and administrative problems. A whole new fresh approach to tariffs and offerings must be found, else data communications will become hopelessly mired in a jungle of undue legality and undue control. I cannot believe that this is the objective of the FCC for such complexity works to their disadvantage as well as the user. The involvement of the state commissions in the area of control only acts to increase the complexity, especially in an area like data communications which, because of technical considerations, makes control even tougher. It would appear that the

four basic determinants for pricing should be 1) switched vs. non-switched services, 2) band width (baud rate) of the facility, 3) volume or number of such facilities employed between points by the user as it impacts on projected (mature offering vs. introductory offering) costs to the carrier and 4) usage (economics of time actually dedicated to the user per day).

If such factors could be the determinants, then the question of whether or not WATS, WADS or both should be supplied, whether there must be a single line offering or a TELPAK offering (or any combination thereof), whether the facility handles data, voice, pictures or noise, or whether the user is politically privileged or not, all become academic. The problem of supplying a new baud rate then would resolve itself down to the engineering job as it should be. The user would then have a firm basis for feasibility determination thru time without having to concern himself with the political acceptability or non-acceptability of a tariff.

b. *Switchable vs. Dedicated Facilities.* Both of these facilities have a place to the user. Volume of use is usually the chief determinant. The lease line in the past has tended to serve only the large volume users. Those organizations with many remote points and with very small volumes at each point can be better served by switched facilities rather than the second-choice compromise of party-lines. Here lies a place for a tariff like the WADS tariff. An adaptation of Western Union's Telex system could also benefit users. The Broad Band offering of WU is part of an answer but, because of limited coverage, has not seen the acceptance it deserves.

Because of the complexity of services offered today, only detailed analysis by the user can determine the optimum choice. The use of simulation and LP techniques has yielded significant results. The use of a simulation technique for the optimum configuration of WATS lines by zones and between measured and full-time by a WATS user yielded a 30% or \$6000 per month differential between the optimum configuration and the pure geographical configuration for a given busy condition.

c. *Economic Considerations.* The following points must be kept in mind by the user in arriving at a facilities choice. Simulation techniques as mentioned above have use herein.

- Volume rates offered by some services.
- Economies of full time vs. measured

time facilities plus the optimum combination of the two. The user should also consider the advisability of using measured time lines on a controlled basis (program control thru parameters) to yield acceptable response times during peak periods and periods when facilities could be temporarily out of service.

- Economics of dual use of facilities for both voice and data either on a timed basis or priority basis.
- Compatibility with purpose—renting of a facility that equals the baud rate need.

d. *Reliability.* Communications facilities, like computers, tend to be statistically reliable. The mean time to failure figures look good, but the mean time to repair times can put a company heavily committed to a communications oriented on-line MIS out of business. The user himself can employ many techniques to correct this problem and end up with a system with much more reliability than a non on-line system. He must give considerations to 1) back-up power supplies that include the communications gear, 2) dual or split communications cables into his data center, 3) protection of the center and its gear from fire and other hazards, 4) insist that separate facilities via separate routes are used to connect locations on the MIS network, and 5) build extra capacity into the MIS hardware system.

Many of the above safeguards could be even improved upon if the carriers would start today engineering better back-up into their systems and facilities. They too must become mindful of the critical position into which they put their customers if they cannot guarantee continuity of service even in severe emergency conditions. Here, like in the computer configuration of an MIS system, duality is a key consideration. It is far better to have the system running at half speed 5% of the time with no 100% failures than to have the system down 2½% of the time. A MIS system can handle the key priority business, if properly designed, even when running at half speed.

e. *Geographical Availability.* Not all facilities are available in all parts of the country. Considerations of international availability of certain facilities are involved in the selection of international headquarters and plant sites in the future.

f. *Equivalent Equipment and Facilities.* Many offerings of both major common carriers are very much

alike. Considerations of service, uptime, physical condition of plant and compatibility of the service with other services must be considered and fitted to the needs of the user.

*Software Considerations—
Hardware Considerations*

To the communications novice, it would appear that the type of facility employed should only be a hardware consideration in that the user need only apply the proper buffer yielding the proper bit rate and right number of bits per character. True, these hardware considerations are important, otherwise the system cannot even function at all over a facility. However, for effective utilization of the facility a fair amount of operating diagnostic software must be employed. This software is used for:

a. *Error Control and Detection.* The user must be aware that the system operates mainly in an unattended mode at the computer site. Humans are not available to scan the incoming or outgoing traffic to detect error conditions. These must be detected by the program. For instance, the repetition of a single character over a given number of times is an indication of a possible stuck tape at the terminal. The computer thus must perform the same checks performed by a human.

b. *Adaptation to a Facility.* Each communications facility has its own peculiarities and vagaries. A perfect facility never actually exists. A dial system has a given number of misdials per thousand trials. We've all detected these in our using the phone system. Sure the system has erred. A dial system that yielded 100% perfection would be too costly for us to afford to use. As such it becomes one of the functions of software to detect misdials and in so doing it must actually log the misdials by location and report these on a periodic basis only when the misdial rate exceeds a standard norm for that facility.

c. *Handling of Special Conditions Peculiar to the Facility.* Certain facilities have conditions attached to their use. These must be controlled by the program. For instance in the utilization of WATS facilities the program can be made to take advantage of idle WATS lines of zones beyond the point to be contacted. Thus if a busy condition exists on all zone 3 lines the computer can, under control, use an open zone 4 line. However, it must first test to be sure that no zone 4 calls are waiting. Similar

logic can also be used to call into play the more expensive measured time lines by using response time indicators as branch parameters.

d. *Trouble Shooting, Analysis and Diagnosis.* Proper diagnostic routines can determine 1) when a line is out of order, 2) when it is failing repeatedly, and 3) whether a problem exists in a line, a subset or a dialer. Sophisticated routines can also determine which particular lead on a subset or dialer is giving us trouble. The reader might ask, "But aren't these the common carrier's problems, why doesn't he take care of them?" Yes, they are the carrier's problems, but you are the user. He cannot afford to have a maintenance man monitor each and every line and piece of equipment day and night, nor can you, the user. But, your computer can. In doing so it can detect problems soon after they start occurring. The system can even be programmed to construct and send an error or trouble message to the carrier's test center indicating the possible location of the trouble. Thus in helping the carrier, the user helps himself keep his facilities at maximum efficiency.

SYSTEM DESIGN CONSIDERATIONS OF THE USER

This paper has covered in some detail thoughts and ideas relative to present and future uses and needs in the physical equipment and facilities areas. There is a whole other area of considerations that the MIS systems designer must be aware of. For the sake of brevity, the writer has chosen to list these considerations since the paper would be incomplete without them. These considerations are of sufficient importance and complexity to be the subject for a future paper in themselves.

SPEED

Considerations

- Cost of terminals and facilities increase with speed but at a declining rate.
- The error rate increases with speed as the transmission approaches the baud or equipment design limits.

Alternatives to Speed

- Exception transmission—management by exception.

- Time sharing with central computation and central data base. (The actual amount of raw new data entered into any computer for the first time each month is very small. It only gets large because of the multiple times we force ourselves to rehandle it. If this can be kept internal, the actual input and output volumes are actually small.)
- Use of more slower terminals dedicated to do a specific job or service a specific functional area.
- The use of time and band multiplexing of lines to yield equal costs for many small slower terminals vs. one larger and faster terminal. In many applications people will experience a faster turn-around and response on the slower devices at their fingertips than from faster devices at a somewhat remote location.

ERROR RATES

Accuracy and Error Control

The two aspects are 1) the potential accuracy problems inherent in equipment and 2) the error situations generated by the humans.

Methods for Control of Mechanical Errors

- Use of data grade facilities.
- Proper hardware design with error detection designed into it.
- Use of hardware that has been engineered with a low error rate in mind.
- Use of block parity checks on medium and high speed terminal equipment.
- Use of equipment with automatic retransmit capabilities.
- Use of program checks to detect format, range and A/N errors.
- The system must be designed to absolutely reject an error transmission at inception.

Methods for Control of Human Errors

- Training of personnel in error control techniques.
- Use of data control and data audit sections.

- Program checks to detect:
 - Format errors
 - Range errors
 - Alpha/Numeric errors
 - Logic errors—by ref. to data base
 - Base data errors—by ref. to data base

DATA SECURITY

Data Security Outside the Organization

Possible Methods:

- Answer back validation
- Phone connection verification (needs to be developed by common carrier)
- Use of pass words
- Computer hang-up and redial

Data Security Inside the Organization

Possible Methods:

- Answer back validation
- Key entry
- Pass words
- Hang-up and redial
- Library procedures and coding
- Data base procedures and coding

RESPONSE TIMES

Considerations:

- Cost—hardware vs. value of time of personnel

- Actual response needed vs. response desired
- Advantages of a reactive system in the efficiency of personnel
- The type of response as contrasted with the capability of the terminal to turn-around quickly
- Time on the line (efficiency of line usage)
- Scheduling and efficiency of the CPU

DATA REDUNDANCY

Considerations:

- Line time used for redundant transmissions
- Time of personnel wasted during redundant transmissions
- Considerations of error rate. Redundancy increases the chance for error
- Redundancy can be materially reduced with on-line communication oriented data base systems
- Considerations of wasted terminal time
- Many terminals designed for redundant operation have a higher cost than other terminals
- Simple terminals tend to have lower error rates than complex terminals.

ELEMENTARY TELEPHONE SWITCHING THEORY APPLIED TO THE DESIGN OF MESSAGE SWITCHING SYSTEMS

Leon Stambler

*RCA, Communication Systems Division
New York City*

INTRODUCTION

The application of telephone techniques to a message switched system will be discussed in this paper.

The first part of the paper will briefly review the traffic techniques employed. Then a model of the message switch is systematically developed based on the traffic analysis approach.

This problem is similar in nature to those encountered in connection with logistics, air and highway traffic control, and phases of military strategy.

REVIEW OF TELEPHONE TRAFFIC TECHNIQUES

Throwdown Studies

One of the techniques which has found use in telephone traffic analysis is the throwdown study. This technique is concerned with large quantities of input data which are statistical in nature.

In this technique, a sufficient number of typical situations are tried to obtain statistically reliable results.

The name "throwdown" stems from the use of dice in the early study of telephone traffic problems. Each die is designated to represent a particular independent event and the faces of the die are designated according to the probability of the event taking

place. By repeatedly "throwing down" a number of such dice and observing the results, the probability of a particular combination of events taking place can be estimated.

Other similar methods based on selections from lists of random numbers have been used in switching traffic studies for a number of years. Mathematicians, using digital computers, have employed similar statistical methods on problems relating to diffusion of gases, electron ballistics and the solution to certain types of differential equations. They have called this the Monte Carlo method.

The throwdown study involves a method of artificially generating traffic, in which an attempt is made to simulate the traffic which would actually appear to the live operational system. This involves processing a large number of calls systematically and determining what happens to each of the calls (i.e., whether they are blocked—lost—or delayed, and for how long as a function of the switching system parameters such as number of trunks, number of registers, processing (or holding) time of common equipment, type of switching network, transmission characteristics, and variability of human or machine inputs.

A single throwdown test will indicate the performance of the system under a specific set of con-

ditions. In a typical traffic study, a given traffic load is first assumed, and a simulated system model to handle this load is devised. The test run will then show the performance of the system under these particular conditions and indicate the adequacy of the initial model and possible improvements. To obtain a proper balance between equipment quantities and traffic load may require several additional runs.

Characteristics of Subscribers Affecting Traffic Data

Traffic presented to a switching system by subscribers is only moderately influenced by the type of system serving their telephones.

Subscribers, although they are individuals, exhibit many "group characteristics," dictated not by the requirements of the switching system but by their mode of life. This fact allows statistical treatment of many observed action distributions without introduction of significant error. However, these group actions also present problems of congestion in switching systems which require detailed throwdown study for solution.

As an example of group characteristic, subscribers do not originate a steady barrage of calls over the 24 hours of the day. During midmorning and mid-afternoon hours traffic is built to a peak value, whereas during certain of the remaining hours it is reduced to a minimum. In some residential areas peak traffic may also occur during the early evening. The traffic analysis however is primarily concerned with the busy hour, the hour in which the greatest number of calls are originated, regardless of its actual time of day occurrence.

Useful datum obtained from busy hour field observations is the calling rate per subscriber (calls per hour) which can be used to set up traffic load conditions on the simulated switching system. The calling rate characteristics can be measured as average calls per hour placed by subscribers in a number of group classifications.

Subscribers, in originating calls, act independently within their classified group in maintaining the average calling rate. Originating times of calls, therefore, occur at random within the hour.

When calls are completed to other subscribers, the connections will be held for a varying amount of time. It has been determined from field observations that the frequency distribution of these holding times is closely approximated by an exponential distribution.

Procedure for a Throwdown Study

The procedure in a throwdown study is to first obtain data representative of the traffic to be handled by the system.

Throwdown input data for subscriber traffic is concerned with expected busy hour traffic (or how many calls should originate in each hour), at what time during each hour will each call be originated, and how long will each call last (holding time).

Specific calls using these factors may be generated artificially with the use of random numbers.

Since subscribers act independently, originating calls will occur at random within each hour.

Throwdown input data representing subscriber originating time can be produced by assigning to each call, of the total number of calls within a specific hour to be studied, a six-digit number from a list of random numbers. The hour is then divided into one million parts, and the assigned random numbers determine the millionth part of the hour during which the call will originate.

(Random numbers can be obtained from a subscriber telephone directory, omitting all numbers that could not be considered of a chance nature, or from the use of a table such as Tippet's *Table of Random Numbers*).

For throwdown purposes, a simplifying assumption can be made: that the holding time of a call (length of the call) is not a continuous variable, but is quantized, so that a particular holding time can have several values. To determine these values an exponential distribution having the proper average is plotted as shown in Fig. 1.

The area under the curve is then divided into

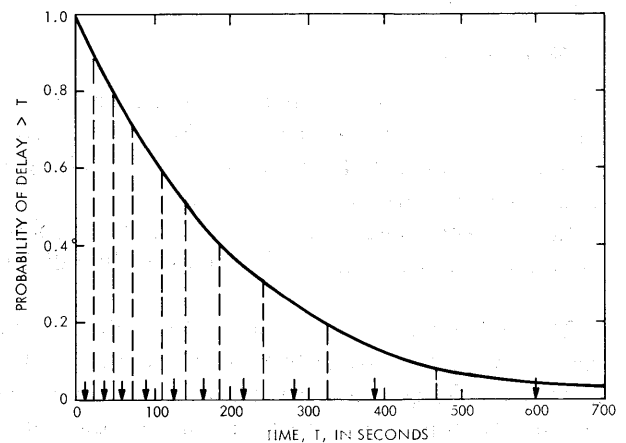


Figure 1. Distribution of holding times.

equal subareas (10 in this case). The mean value of holding time of each subarea is then used to represent each subarea. Ten holding times are thus produced, which are weighted according to the exponential distribution.

These holding times can be designated with numbers 0 to 9, and then assigned to random calls by choosing single digit random numbers from a random number table.

As a simple example of how the throwdown is used suppose that it is desired to determine how often on the average an "all trunks busy" condition will occur in a particular group of trunks handling inter-office calls.

With the data of call origination times and holding times prepared, the throwdown run can be started. The calls are listed in the order of their originating times. The first call is assigned to the first idle trunk. A record that this trunk is busy is made and the time at which it will become idle determined by adding the assigned holding time to the time of origination. This is also recorded. The call which follows in time of origination is then assigned to the next idle trunk and the process continued for succeeding calls. Before each call is established the release times of all busy trunks are scanned to determine whether any busy trunk should be made idle. In setting up each call, idle trunks are chosen from the group in the same order of preference that would be used in the system being simulated.

Thus, the performance of an actual system is reproduced with considerable accuracy and detailed records of this performance can be made. From a study of these records the desired information can be determined. The probability of encountering an "all trunks busy" condition can be found, the average number of trunks busy can be determined, or a frequency distribution chart showing the percentage of the time the number of busy trunks is above any given number can be constructed. If proper records are kept such information as the average number of trunks searched over in locating an idle trunk can be determined.

This particular problem can also be solved by analytical methods and is presented here only to illustrate the application of the throwdown technique.

Lost Calls Served on a Blocked or Delayed Basis

Another analytical method which has been used for many years is the lost or delayed call analysis.

If a telephone subscriber does not receive dial tone from a register immediately after lifting his handset, the call is not considered to be lost, but is delayed, because if he waits he will get the register and receive dial tone. Naturally, the object is to make this delay as short as possible, consistent with providing an adequate grade of service.

Calls which are delayed can either be served in a random order (according to random theory) or stored in a queue and served on a first come first served basis (in order of arrival queue theory).

The problem of traffic engineering on a delay basis has been studied by A. K. Erlang,¹ E. C. Molina,² F. Pollaczek³ and C. D. Crommelin,⁴ among others.

The usual way of describing delay distribution is to indicate the probability that a call will be delayed in excess of a certain time (generally expressed in multiples of the average holding time).

It has been shown that calls with exponential holding times served in random order yield greater delays than calls served in order of arrival.

For calls with constant holding time an analysis was given by Pollaczek³ in 1930, which provides the formula which gives directly the probability of a delay greater than *t* seconds.

The exponential holding time analysis given by E. C. Molina² in 1927 for calls served in order of arrival gives the probability of a delay greater than zero, and also the probability of a delay greater than *t*.

Figure 2 indicates the formulas which are used in

$$(1) P(>0) = \frac{\frac{a^c e^{-a}}{c!} \cdot \frac{c}{c-a}}{1 - P(c,a) + \frac{a^c e^{-a}}{c!} \cdot \frac{c}{c-a}}$$

$$(2) P(c,a) = \text{Poisson} = \frac{\text{Summation}}{\sum_{x=c}^{\infty} \frac{a^x e^{-a}}{x!}}$$

$$(3) P(>t) = P(>0) e^{-(c-a) \frac{t}{\bar{T}}}$$

$$(4) \bar{d} = P(>0) \frac{\bar{T}}{c-a}$$

$$(5) d = \frac{\bar{d}}{P(>0)} = \frac{\bar{T}}{c-a}$$

where *a* = erlangs of traffic offered
c = number of registers (or outlets)
 \bar{T} = average holding time

$$(6) \alpha = \frac{a}{c} = \text{occupancy (\% of time } c \text{ is being used)}$$

$$(7) a = \text{erlangs} = \frac{\text{No of busy hr. calls} \times \text{ave. holding time in sec.}}{3600 \text{ sec/hr.}}$$

Figure 2. Exponential holding time analysis.

this analysis. Equation (1) in the figure is probability of a delay greater than zero, or the blocking probability. $P(c,a)$ is the blocking probability for the case of an infinite number of subscriber sources and where lost calls are held for one average holding time. Equation (3) is the probability of a delay greater than t seconds. Equation (4) is the average delay on calls, averaged over all calls, and Eq. (5) is the average delay on those calls which have been blocked.

For some specific examples of how these formulas may be applied in practice, see Example No. 1 (Fig. 3).

Example No. 1. Suppose we are required to determine the number of registers in a switching center so that no more than 1% of the busy hour calls receive dial tone delays in excess of one second. (The register is a time-shared unit of equipment, commonly used in telephone switches.)

During the busy hour the switching center will be required to handle 1200 newly originated calls. The subscribers use the registers with an exponentially distributed holding time with an average of six seconds.

First we compute the busy hour traffic offered in erlangs as the quantity a . Then we assume a value for c , the number of registers. We then compute $P(c,a)$ or get the value of $P(c,a)$ from a table of cumulative terms of the Poisson Formula. We next compute the blocking probability where the term $a^c e^{-a}/c!$ can be obtained from a table of individual terms of the Poisson formula. We next compute the

Example No. 1

Given: $P(>1 \text{ sec}) = 0.01 = 1\%$
 Busy Hour Traffic } = 1200 calls
 Average Register holding time } = $\bar{T} = 6$ seconds
 Find: The number of registers = c
 Solution: $a = \frac{1200 \text{ calls} \times 6 \text{ sec}}{3600 \text{ sec (busy hr.)}} = 2.0$ erlangs
 Assume: $c = 6$ registers
 $P(c,a) = 0.016$ (from eq (2))
 $P(>0) = \frac{(0.012)(1.5)}{1-0.016 + (0.012)(1.5)} = 0.018$ (from eq (1))
 $P(>1) = (0.018)(e^{-.667}) = 0.9\%$
 Hence: $c = 6$ registers was a correct assumption
 Also: $P(>2) = (0.018)(e^{-1.33}) = 0.5\%$
 And: $\bar{d} = (0.018)(1.5) = 0.027$ second

Figure 3. Register analysis.

Example No. 2

Given: $c = 10$ trunks
 Busy Hour Traffic } = 100 calls
 Average holding time of a call } = 3 minutes
 Find: The percentage of the calls that are delayed.
 Solution: $a = \frac{100 \times 3}{60} = 5$ erlangs
 $c = 10$ trunks
 $P(c,a) = 0.032$
 $P(>0) = \frac{(0.018)(2)}{1-0.032 + 0.036} = 0.035$
 Hence: 3.5% of the busy hour calls will be delayed.

Figure 4. Delay on trunks.

probability of a delay of more than one second. Since our answer is 0.9% the initial assumption of six registers was correct. We can also compute $P(>2)$ and the average delay.

For another example of the application of these formulas, see Example No. 2 (Fig. 4).

Example No. 2. Suppose that a 10-trunk route carrying exponentially distributed holding time calls with an average length of 3 minutes is loaded with 100 calls during the busy hour and any calls delayed will be served in order of arrival. What percentage of these calls will be delayed? First we compute the traffic offered to the trunks in erlangs as the quantity a . We next look up $P(c,a)$ and $a^c e^{-a}/c!$ in tables, from which we can compute the blocking probability $P(>0)$.

We see that 3.5% of the busy hour calls are delayed.

TELEPHONE TRAFFIC MODEL FOR THE MESSAGE SWITCHED SYSTEM

General

In this analysis, one of the criteria to be used in determining the effectiveness of the message switch is the trunk utilization. We will assume that calls or messages from subscriber terminals originate at random. Of these calls, a certain percentage which originate simultaneously may require access to a given trunk. The message switch queues (stores) these calls in a manner which permits maximum occupancy of the trunk.

In developing this analysis, we make the following assumptions:

1. Message lengths have an exponential frequency distribution.

2. Messages which are blocked and delayed for trunk transmission are held in the "In-Transit" store for delivery on the basis of order or arrival.
3. For this analysis we assume a 2400-bit/sec trunk speed.
4. We will assume that the average holding time for a message which is being transmitted over the trunk is 3 seconds.

Additional assumptions will be made and these will be pointed out as the analysis develops.

Blocked Calls Delayed

The next assumption we will make is that we have one trunk ($c=1$), and that it is occupied with message traffic 70% ($\alpha = 0.7$) of the time during the busy hour. From this assumption we can compute the number of messages sent over the trunk during the busy hour.

$$N = \frac{1 \text{ message}}{3 \text{ sec}} \times 3600 \text{ sec/hr} \times 0.7 \text{ occupancy} \\ = 840 \text{ messages/hr}$$

We next desire to compute, with the use of Eq. (5) in Fig. 2, the average delay encountered for those calls which have been blocked from accessing the trunk immediately.

From Eq. (7) in Fig. 2, the traffic submitted to the trunk is:

$$a = \frac{840 \times 3}{3600} = 0.7 \text{ erlangs}$$

and

$$c = 1 \text{ trunk}$$

Hence

$$\bar{d} = \frac{\bar{t}}{c-a} = \frac{3}{1-0.7} = 10 \text{ sec}$$

This means that when the trunk is 70% occupied (during the busy hour), those messages which are delayed will have an average waiting time of 10 seconds before they can get on the trunk. These messages would then wait on the "In-Transit" store.

As indicated above we have shown that 840 messages are sent over the trunk during the busy hour. The next assumption will be to assume that 840 messages are also originated by subscriber terminals and enter the message switch during the busy hour.

Trunk vs Line Transmission Speeds

At this point, two alternatives present themselves, since we must deal with the problem of a transmission speed differential between subscriber lines and the trunk.

To cope with this problem, one alternative is to provide a one-message buffer, on a per line basis to take care of the speed differential. With this approach, when the total message has been received in the line buffer, it is then immediately transferred to the "In-Transit" store, where it may have to wait before going out on the trunk. The other alternative is to provide a one bit buffer on a per line basis to take care of the speed differential. With this approach, when a bit has been received in the line bit store, it is then immediately transferred to the "In-Transit" store, where it may have to wait for the total message to be accumulated, and may have an additional wait before accessing the trunk.

In transferring information between the input buffer and the "In-Transit" store, the alternatives indicated above assume that the access time plus the transfer time from the buffer to the "In-Transit" store is short enough such that no incoming data bits are lost.

In the case where buffered bits are transferred directly to a magnetic drum memory, which is used as an "In-Transit" store, the access time will in general be greater than the input bit transmission time. The minimum size buffer can then be computed by first summing the access time to a selected drum memory location with the time to transfer from the buffer to the "In-Transit" store, and then multiplying this sum by the input transmission rate.

Both of these alternatives will be developed and compared in the analysis that follows. However, the problems associated with message, block, or bit integrity and channel coordination are not covered in this analysis, and remain as areas for future study.

Case I: One Message Buffer per Line. We desire to compute the size of the "In-Transit" store in terms of the number of average message stores. This computation will be based on a probability that one message in a thousand will fail to find an empty message slot available to it when it is ready to be transferred from the line message buffer to the "In-Transit" store.

As previously computed, the average holding time of delayed messages in the "In-Transit" store is 10 seconds.

We now compute the traffic submitted from all line buffers to the "In-Transit" store.

$$a = \frac{840 \text{ messages} \times 10 \text{ sec}}{3600 \text{ sec/hr}} = 2.33 \text{ erlangs}$$

For a blocking probability ($P(>0)$) of .001 for messages sent to the "In-Transit" store from the line buffers, with 2.33 erlangs of traffic submitted, we now use Eq. (1) in Fig. 2 to determine the value of c , which in this case is the number of message stores required. From traffic tables (given $P(>0) = .001$, and $a = 2.33$), we find that $c = 9$ message stores. Hence, the minimum size of the "In-Transit" store required is 9 messages. Another way of looking at this result is to say that the probability of more than 9 messages requiring simultaneous access to the "In-Transit" store from the line buffers is 1 in 1000.

We will next make another assumption with respect to the traffic characteristics of subscriber lines during the busy hour. Assume that all lines are occupied 70% of the time sending message traffic to the switch during the busy hour.

To simplify matters, let us further assume that all lines are TTY's operating at 75 bits/sec.

We next compute the average holding time of a message on a line.

$$\begin{aligned} \bar{t} &= \frac{\text{trunk speed}}{\text{line speed}} \times \text{av message holding time on trunk} \\ &= \frac{2400}{75} \times 3 = 96 \text{ seconds} \end{aligned}$$

We next assume that the number of messages sent by each line is uniformly distributed for all lines. (This is a conservative assumption, since we still insist upon 70% occupancy during the busy hour for all lines.) Hence, the number of messages sent per line during the busy hour is:

$$\begin{aligned} &\frac{\text{busy-hour occupancy} \times 3600 \text{ sec/hr}}{\text{av message holding time on line}} \\ &= \frac{(0.7) (3600)}{96} = 26.25 \text{ messages/line} \end{aligned}$$

Then the number of TTY lines which can be accommodated by this model is:

$$\frac{\text{No. messages submitted by all lines}}{\text{No. messages per line}}$$

$$= \frac{840}{26.25} = 32 \text{ lines}$$

Fig. 5 is a block diagram of the resulting model for Case I.

Case II: One Bit Buffer per Line. We desire to compute the size of the "In-Transit" store in terms of the number of message stores. This computation will be based on a probability that one message in a thousand will fail to find an empty message slot available to it, when the first bit of the message is ready to be transferred from the line bit buffer to the "In-Transit" store.

As previously computed, the average holding time of delayed messages in the "In-Transit" store is 10 seconds. In order not to complicate matters, we will assume that the total message must be accumulated in the "In-Transit" store before any part of it can access the trunk.

Hence, for TTY lines, the average message accumulation time (as previously computed) is 96 seconds.

We now compute the traffic submitted from all line bit buffers to the "In-Transit" store as:

$$\begin{aligned} a &= \frac{(840 \text{ messages/hr.}) (10 + 96) \text{ sec}}{3600 \text{ sec/hr}} \\ &= 24.7 \text{ erlangs} \end{aligned}$$

For a blocking probability ($P(>0)$) of .001 for messages sent to the "In-Transit" store from the line bit buffers, with $a = 24.7$ erlangs of traffic submitted, we again use Eq. (1) of Fig. 2 to determine the value of c , which in this case is the number of

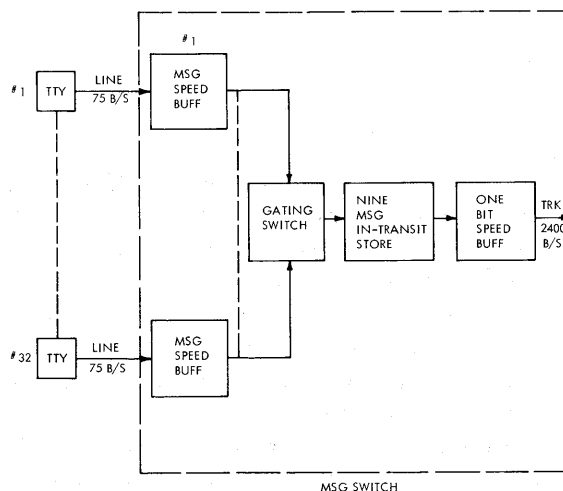


Figure 5. Model for Case I.

message stores required. From traffic tables (given $P(>0) = .001$, and $a = 24.7$), we find the $c = 41$ message stores. Hence, the size of the "In-Transit" store required to handle 24.7 erlangs of input traffic is 41 messages.

For a 70% line occupancy during the busy hour, the optimum of TTY lines is again 32. Figure 6 is a block diagram of the resulting model for Case II. It is also interesting to note that a 41-message "In-Transit" store can also accommodate eight 2400-bit/sec input lines, when these lines deliver a total busy hour traffic of 24.7 erlangs.

Other Types of Traffic Flows

Thus far the model which has been constructed has been very simpleminded, i.e., the only traffic considered has been that which was originated by many subscriber lines and forwarded over a single trunk.

It is now desired to take a broader look at the traffic flow, so as to arrive at a more realistic and versatile model which will accommodate outgoing trunk traffic to more than one switch, incoming trunk traffic from more than one switch, and outgoing local line traffic.

For the case of outgoing traffic to lines, and trunks to more than one switch, one may choose to consider the size of the "In-Transit" store as a fixed system parameter, in which case additional outgoing trunks and lines provide additional outlets for messages accumulated in the "In-Transit" store. However, if additional outlets are added to a fixed capacity "In-Transit" store, then trunk and line occupancies are sacrificed. That is, all lines and trunks may be idle

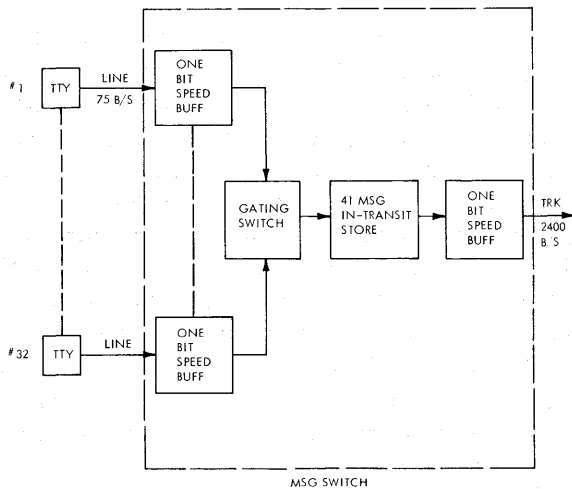


Figure 6. Model for Case II.

part of the time and may not be gainfully employed during the busy hour.

GENERALIZED ANALYSIS

- Let C_0 = the number of outgoing trunks
- C_i = the number of incoming trunks
- L_0 = the number of outgoing lines
- L_1 = the number of incoming lines
- α_{c_0} = the busy hour occupancy of all outgoing trunks
- α_{c_i} = the busy hour occupancy of all incoming trunks
- α_{l_0} = the busy hour occupancy of all outgoing lines
- α_{l_1} = the busy hour occupancy of all incoming lines
- t_c = average message holding time on a trunk
- t_l = average message holding time on a line

First consider all outgoing traffic.

The number of outgoing messages during the busy hour for all outgoing trunks is:

$$n_{c_0} = (\alpha_{c_0}) \frac{1 \text{ message}}{t_c \text{ sec}} (C_0 \text{ trunks}) \times 3600 \frac{\text{sec}}{\text{hr}}$$

$$= \frac{\alpha_{c_0} \times C_0 \times 3600}{t_c}$$

The number of outgoing messages during the busy hour for all lines is:

$$n_{l_0} = (\alpha_{l_0}) \frac{1 \text{ message}}{t_L \text{ sec}} (L_0 \text{ lines}) \times 3600 \frac{\text{sec}}{\text{hr}}$$

$$= \frac{\alpha_{l_0} \times L_0 \times 3600}{t_L}$$

The outgoing trunk traffic in erlangs is:

$$A_{c_0} = \frac{(\alpha_{c_0} \times C_0 \times 3600) (\bar{t}_c)}{3600} = \alpha_{c_0} \times C_0 \text{ erlangs}$$

The outgoing line traffic in erlangs is:

$$A_{l_0} = \frac{(\alpha_{l_0} \times L_0 \times 3600) (\bar{t}_L)}{3600} = \alpha_{l_0} \times L_0 \text{ erlangs}$$

The average delay for outgoing trunk messages which have been blocked is:

$$\bar{d}_{c_0} = \frac{\bar{t}_c}{C_0 - A_{c_0}} = \frac{\bar{t}_c}{C_0 (1 - \alpha_{c_0})}$$

The average delay for outgoing line messages which have been blocked is:

$$\bar{d}_{L_0} = \frac{\bar{t}_L}{L_0 - A_{L_0}} = \frac{\bar{t}_L}{L_0 (1 - \alpha_{L_0})}$$

Next, consider all incoming traffic. The number of incoming messages during the busy hour for all trunks is:

$$\begin{aligned} n_{c_i} &= (\alpha_{c_i}) \left(\frac{1 \text{ message}}{\bar{t}_c \text{ sec}} \right) (C_i \text{ trunks}) \left(\frac{3600 \text{ sec}}{\text{hr}} \right) \\ &= \frac{\alpha_{c_i} \times C_i \times 3600}{\bar{t}_c} \end{aligned}$$

The number of incoming messages during the busy hour for all lines is:

$$\begin{aligned} n_{L_i} &= (\alpha_{L_i}) \left(\frac{1 \text{ message}}{\bar{t}_L \text{ sec}} \right) (L_i \text{ lines}) \left(\frac{3600 \text{ sec}}{\text{hr}} \right) \\ &= \frac{\alpha_{L_i} \times L_i \times 3600}{\bar{t}_L} \end{aligned}$$

We will examine two cases. One provides speed buffering between slow-speed lines and high-speed trunks using message stores on lines and trunks. The other case provides one-bit stores on lines and trunks.

Case I: Speed buffering with message stores for each line and for each trunk.

The incoming trunk traffic submitted to the "In-Transit" store is:

$$\begin{aligned} A_{c_i} &= \frac{(n_{c_i}) (d_{c_0})}{3600} \\ &= \frac{\left(\frac{\alpha_{c_i} \times C_i \times 3600}{\bar{t}_c} \right) \left(\frac{\bar{t}_c}{C_0 (1 - \alpha_{c_0})} \right)}{3600} \\ &= \left(\frac{\alpha_{c_i}}{1 - \alpha_{c_0}} \right) \left(\frac{C_i}{C_0} \right) \text{ erlangs} \end{aligned}$$

The incoming line traffic submitted to the "In-Transit" store is:

$$\begin{aligned} A_{L_i} &= \frac{(n_{L_i}) (d_{L_0})}{3600} \\ &= \frac{\left(\frac{\alpha_{L_i} \times L_i \times 3600}{\bar{t}_L} \right) \left(\frac{\bar{t}_L}{L_0 (1 - \alpha_{L_0})} \right)}{3600} \\ &= \left(\frac{\alpha_{L_i}}{1 - \alpha_{L_0}} \right) \left(\frac{L_i}{L_0} \right) \text{ erlangs} \end{aligned}$$

The total input traffic submitted to the "In-Transit" store from line message buffers and trunk message buffers is:

$$A_{i_n} = \left(\frac{\alpha_{c_i}}{1 - \alpha_{c_0}} \right) \left(\frac{C_i}{C_0} \right) + \left(\frac{\alpha_{L_i}}{1 - \alpha_{L_0}} \right) \left(\frac{L_i}{L_0} \right) \text{ erlangs}$$

This input traffic can now be used to determine the size of the "In-Transit" store which is required for a blocking probability ($P (>0)$) of .001. For example, suppose, $C_i = C_0$ and $L_i = L_0$. (This is the case of a full-duplex system.)

$$A_{i_n} = \frac{\alpha_{c_i}}{1 - \alpha_{c_0}} + \frac{\alpha_{L_i}}{1 - \alpha_{L_0}} \text{ erlangs for full duplex.}$$

Suppose now that $\alpha_{c_i} = \alpha_{c_0} = \alpha_{L_i} = \alpha_{L_0} = 0.8$; that is, all inputs and outputs are occupied with message traffic for 80% of the busy hour. Then,

$$A_{i_n} = \frac{0.8}{1 - 0.8} + \frac{0.8}{1 - 0.8} = \frac{1.6}{0.2} + 8 \text{ erlangs}$$

Then for $P (>0) = .001$, using Equation (3) in Fig. 2, the number of message stores required in the "In-Transit" store is 19 message stores. Figure 7 shows a plot of occupancy vs size of the "In-Transit" store. Two curves are indicated. One curve shows the size of the store for equal occupancies on all incoming and outgoing lines and trunks. For this curve, when either all outgoing trunks or lines are 100% occupied, no additional incoming traffic can get out. Further, any additional incoming traffic (α_{c_i} or $\alpha_{L_i} > 0$) will be blocked from entering the "In-Transit" store. In such case, the additional incoming traffic must either wait at the source, or overflow to some other place.

The other curve shows the size of the store required for normal plus multiaddress traffic. For this

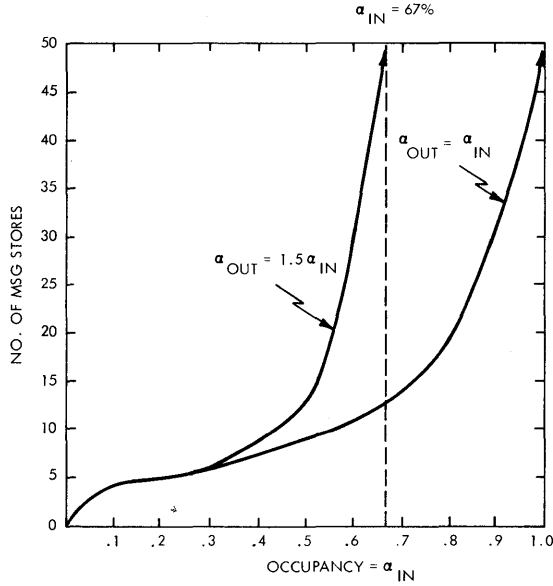


Figure 7. Size of store vs occupancy—Case I.

case we have assumed that the occupancy of outgoing lines and trunks is one and one-half times as great as that of incoming lines and trunks. For this case,

$$\alpha_{c_0} = 1.5 \alpha_{c_i} \text{ and } \alpha_{1_0} = 1.5 \alpha_{1_i}$$

so

$$A_{i_n} = \frac{\alpha_{c_i}}{1 - \frac{3\alpha_{c_i}}{2}} + \frac{\alpha_{1_i}}{1 - \frac{3\alpha_{1_i}}{2}} \text{ erlangs.}$$

For this case, when either all incoming lines or all trunks have an occupancy of 67%, the output lines or trunks will be 100% occupied. Any additional traffic or occupancy greater than 67% on all lines or all trunks will cause this traffic to be blocked from entering the “In-Transit” store. These messages will have to wait at the source or overflow somewhere else.

Case II: All messages are speed buffered on the basis of bit buffer stores for each line and trunk.

In this case, when a bit has been received in the line or trunk bit store, it is then immediately transferred to the “In-Transit” store, where it waits for the total message to be accumulated, and then the message may have an additional waiting time in the “In-Transit” store before it can access an outgoing line or trunk.

The incoming trunk traffic submitted to the “In-Transit” store is:

$$\begin{aligned} A_{c_i} &= \frac{(N_{c_i})(\bar{d}_{c_0} + \bar{t}_c)}{3600} \\ &= \frac{\left(\frac{\alpha_{c_i} \times C_i \times 3600}{\bar{t}_c}\right) \left(\frac{\bar{t}_c}{C_0(1 - \alpha_{c_0})} + \bar{t}_c\right)}{3600} \\ &= \left(\frac{\alpha_{c_i}}{1 - \alpha_{c_0}}\right) \left(\frac{C_i}{C_0}\right) + \alpha_{c_i} C_i \text{ erlangs} \end{aligned}$$

The incoming line traffic submitted to the “In-Transit” store is:

$$\begin{aligned} A_{1_i} &= \frac{(N_{1_i})(\bar{d}_{1_0} + \bar{t}_L)}{3600} \\ &= \frac{\left(\frac{\alpha_{1_i} \times L_i \times 3600}{\bar{t}_1}\right) \left(\frac{\bar{t}_L}{L_0(1 - \alpha_{1_0})} + \bar{t}_L\right)}{3600} \\ &= \left(\frac{\alpha_{1_i}}{1 - \alpha_{1_0}}\right) \left(\frac{L_i}{L_0}\right) + \alpha_{1_i} L_i \text{ erlangs} \end{aligned}$$

The total input traffic submitted to the “In-Transit” store from the line and trunk bit buffers is:

$$\begin{aligned} A_{i_n} &= \left(\frac{\alpha_{c_i}}{1 - \alpha_{c_0}}\right) \left(\frac{C_i}{C_0}\right) + \alpha_{c_i} C_i \\ &+ \left(\frac{\alpha_{1_i}}{1 - \alpha_{1_0}}\right) \left(\frac{L_i}{L_0}\right) + \alpha_{1_i} L_i \text{ erlangs} \end{aligned}$$

This input traffic can now be used to determine the size of the “In-Transit” store which is required for a blocking probability ($P(>0)$) of .001. For the case of a full duplex system $C_i = C_0$ and $L_i = L_0$, so

$$\begin{aligned} A_{i_n} &= \frac{\alpha_{c_i}}{1 - \alpha_{c_0}} + \frac{\alpha_{1_i}}{1 - \alpha_{1_0}} \\ &+ (\alpha_{c_i} C_i) + (\alpha_{1_i} L_i) \text{ erlangs} \end{aligned}$$

In this case we note from the above input traffic that the size of the “In-Transit” store will depend on the occupancies and, in addition, on the number of incoming lines and trunks. Figure 8 shows a plot of occupancy vs size of the “In-Transit” store for equal values of occupancy on lines and trunks, and for

several values of $(C_i + L_i)$ as a parameter. For this case

$$A_{i_n} = \left(\frac{\alpha}{1-\alpha} \right) 2 + \alpha(C_i + L_i)$$

Also shown on Fig. 8 is the total number of messages stores for Case I. This is obtained by taking the store size curve in Fig. 7, for equal occupancies, and adding to it the number of message stores required for speed buffering, which is a function of $C + L$. The intersections of these curves on Fig. 8 provides a method for comparing the two cases for the equal occupancy condition. These curves intersect in the region of 75 to 80% occupancy (for equal numbers of terminations). Hence, one can conclude that for high values of occupancy (constant high traffic periods), both approaches give about the same service for about same cost. For low occupancy, Case II is far superior, since per line message buffers are not used.

Figure 9 shows a plot similar to Fig. 8. However, these curves are plotted for normal + multiaddress traffic. For this case

$$\alpha_{c_0} = 1.5 \alpha_{c_i} \text{ and } \alpha_{l_0} = 1.5 \alpha_{L_i}$$

so

$$A_{i_n} = \frac{\alpha_{c_i}}{1-3 \alpha_{c_i}} + \frac{L_i}{1-3 \alpha_{L_i}} + \alpha_{c_i} C_i + \alpha_{L_i} L_i$$

The curves intersect in the region of 46–58% input occupancy. Hence, for inputs below 50% occupancy,

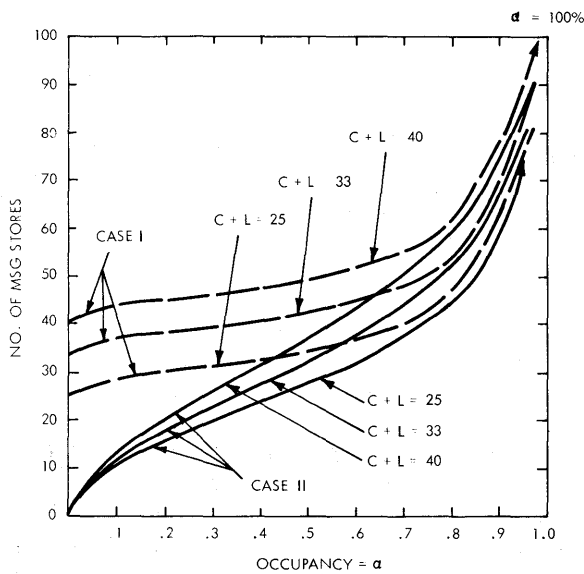


Figure 8. Size of store vs occupancy—Case II.

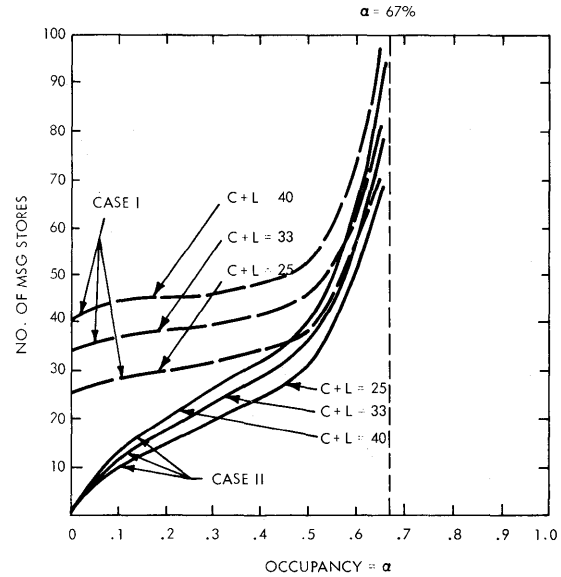


Figure 9. Cases I and II for multiple address.

Case II is preferred, and for inputs above 50% occupancy, the two cases are very similar.

A FUNCTIONAL RELATIONSHIP BETWEEN THE NUMBER OF LINES AND TRUNKS

The quantity of lines and trunks that are to be served by a given switch can be related by the following variables:

1. The quantity of traffic flow between lines and trunks. This can be stated in terms of their respective occupancies, and a multiple address factor.
2. The type of transmission facility which is provided on lines and trunks. These can be related by comparing the length of time required to transmit an average message on each.

As before, we can compute the total number of messages received by the switch, during the busy hour, from all lines and trunks as follows:

$$I = \frac{\alpha_{c_i} \times 3600 \times C_i}{\bar{t}_c} + \frac{\alpha_{L_i} \times 3600 \times L_i}{\bar{t}_L}$$

We next compute the total number of messages transmitted from the switch, during the busy hour, from all lines and trunks as follows:

$$O = \frac{\alpha_{c_0} \times 3600 \times C_0}{\bar{t}_c} + \frac{\alpha_{L_0} \times 3600 \times L_0}{\bar{t}_L}$$

Each input message may generate one or more output messages. Hence, the total number of input messages may be related to the total number of output messages by some averaged constant, K .

We categorize this averaged constant K as the multiaddress constant. Hence, $0 = K I$ or

$$\frac{\alpha_{c_0} \times 3600 \times C_0}{\bar{t}_c} + \frac{\alpha_{L_0} \times 3600 \times L_0}{\bar{t}_L} \\ = K \frac{\alpha_{c_i} \times 3600 \times C_i}{\bar{t}_c} + \frac{\alpha_{L_i} \times 3600 \times L_i}{\bar{t}_L}$$

This reduces to:

$$\frac{\alpha_{c_0} \times C_0 - K \alpha_{c_i} C_i}{\bar{t}_c} = \frac{K \alpha_{L_i} L_i - \alpha_{L_0} L_0}{\bar{t}_L}$$

For the full duplex system $L_i = L_0 = L$, and $C_i = C_0 = c$. Hence,

$$\frac{L}{C} = \frac{\bar{t}_L}{\bar{t}_c} \cdot \left| \frac{\alpha_{c_0} - K \alpha_{c_i}}{K \alpha_{L_i} - \alpha_{L_0}} \right|$$

We can illustrate the use of this formula with a simple example. Suppose all traffic was from incoming lines to outgoing trunks, i.e.,

$$\alpha_{c_i} = \alpha_{L_0} = 0$$

Then

$$\frac{L}{C} = \frac{\bar{t}_L}{\bar{t}_c} \cdot \frac{\alpha_{c_0}}{K \alpha_{L_i}}$$

If $\alpha_{c_0} = \alpha_{L_i} = 1.0$, and $K = 1.5$, and $\bar{t}_L = 96$ sec and $\bar{t}_c = 3$ sec, then

$$\frac{L}{C} = \frac{96}{3} \cdot \frac{1}{1.5} = 21$$

For a system that has already been constructed and is in use, the ratio of L/C has been fixed by the equipment. The holding times are somewhat fixed by the transmission speed facilities, but these also depend on the statistical nature of the length of messages. The occupancies and the multiaddress factor K are the true variables in this case, since these are determined by the actual traffic.

Since the parameters on the right-hand side of this formula all vary under operational traffic condi-

tions, both sides of the identity may not be equal, in which case the switch is not performing as well as it might be.

If the system is designed to force the identity to be true, then it is operating with maximum efficiency. All parameters on the right-hand side are measurable and can be computed by the switch as it passes traffic. The switch can then be conceived to be adaptive in nature, and when it finds that the identity is not true, it can automatically exercise line or trunk load control to readjust the traffic flow.

CONCLUSION

The traffic techniques described herein have been used by system engineers for telephone trunking problems since the early 1900's. The equations and results given are valid for message switching, providing the statistics of the message traffic agree reasonably with the assumption of exponential message length distribution.

There still remains a host of problems concerned with channel coordination procedures and hardware-software tradeoffs that can be studied using the techniques described in this paper.

REFERENCES

1. A. K. Erlang, "Solution of Some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges," *P.O.E.E. Jl.*, vol. 10, p. 189 (1917).
2. E. C. Molina, "Application of the Theory of Probability to Telephone Trunking Problems," *B.S.T. Jl.*, vol. VI, July 1927.
3. F. Pollaczek, various articles including those found in: *Mathematische Zetschrift*, vol. 32, pp. 64-100 and 729-50 (1930); *Electrische Nachrichten—Technik*, June and July 1931; *Telegraphen und Fernsprech—Technik*, 1930, pp. 71-78.
4. C. D. Crommelin, "Delay Probability Formulae, When the Holding Times are Constant," *P.O.E.E. Jl.*, vol. 25, p. 41; "Delay Probability Formulae," *ibid.*, vol. 26, p. 266.
5. R. I. Wilkinson, "Working Curves for Delayed Exponential Calls Served in Random Order," *B.S.T. Jl.*, vol. 32, pp. 360-83 (Mar. 1953).

TOWARD A COOPERATIVE NETWORK OF TIME-SHARED COMPUTERS

Thomas Marill

*Computer Corporation of America, Cambridge,
Massachusetts*

and

Lawrence G. Roberts

MIT, Lincoln Laboratory, Lexington, Massachusetts*

INTRODUCTION: NETWORKS AND THE PROBLEM OF COMPUTER INCOMPATIBILITY

Incompatible machines represent an old problem in the computer field. Very often, because of computer incompatibility, programs developed at one installation are not available to users of other installations. The same program may therefore have to be rewritten dozens of times.

Assume, for example, that a program which performs a syntactic analysis of natural English sentences has been developed for a certain computer, Y. Such a program would be of interest to workers in the fields of natural-language inputs to computers, mechanical translation and linguistics, information retrieval, command and control, and a number of other ancillary disciplines. Unfortunately, the program would be available only to those who had direct access to Y (or to a computer compatible with Y);

* This work was supported in part under a subcontract of MIT, Lincoln Laboratory, which is operated with support from the Advanced Research Projects Agency.

users of other installations would have to recode the program for their own computers, at a cost per computer comparable to the cost of the original programming development. Viewed on a nationwide scale, such inefficiencies can be enormously expensive.

The time-honored remedies for computer incompatibility have been the following.

1. Use identical computers. Thus, for example, the use of two identically modified IBM 7094's at Project MAC and at the MIT Computation Center made possible the development of the Compatible Time-Sharing System (CTSS), allowing programs written for one system to be run on the other.
2. Write programs in a high-level language for which compilers exist on different machines. Thus, a given FORTRAN source program can be compiled for and run on a large number of different computers.

Unfortunately, these remedies have worked quite badly in the past and will probably work as badly in future time-sharing environments. Regarding remedy (1), there is no indication that the proliferation of hardware is ending. It is not difficult to find examples of situations where, even within a single organization, incompatible new computers are being added to existing ones, requiring duplication of programming efforts. Regarding remedy (2), one finds that new computer languages are developed daily. It has been estimated that the number of time-sharing installations is roughly equal to the number of languages offered among them.

Thus we see no particular reason to believe that the old remedies will work better in the future than in the past. The possibility exists, however, that the technique of computer networking will contribute to the solution of the problem. Since time-sharing systems, by their nature, are designed to be operated remotely on a real-time basis, we can envision the possibility of the various time-shared computers communicating directly with one another, as well as with their users, so as to cooperate on the solution of problems. To return to our earlier example, if a user of machine X wanted to use, as part of a larger program, the syntactic-analysis routine running on computer Y, this larger program would communicate the sentence to be analyzed to Y, cause the syntactic-analysis program to run on Y, accept the results, and go on running at X.

Such an approach would circumvent the problem of incompatible machines by allowing all programs to run on their home computer.

A program which ran on computer Y would not need to be rewritten or recompiled for computer X in order to be part of a system running on X; the program would run on Y, as always, and its output would be communicated to X at the proper time.

Within a computer network, a user of any cooperating installation would have access to programs running at other cooperating installations, even though the programs were written in different languages for different computers. This forms the principal motivation for considering the implementation of a network.

The establishment of a network may lead to a certain amount of specialization among the cooperating installations. If a given installation, X, by reason of special software or hardware, is particularly adept at matrix inversion, for example, one may expect that users at other installations in the network

will exploit this capability by inverting their matrices at X in preference to doing so on their home computers.

Carrying this train of thought one step further, it may develop that small time-shared computers will be found to be efficiently utilized when employed only as communication equipment for relaying the users' requests to some larger remote machine on which the substantive work is done. Such smaller installations might then be considered to be "retail outlets" for the "wholesale computer power" provided by the giant machines.*

SOFTWARE CONSIDERATIONS

The Elementary Approach

We will first discuss an elementary approach to the problem of forming computer networks. This approach certainly does not represent the best alternative, but it has the advantage as a point of departure of being by far the simplest, since it allows a network of existing time-sharing systems to be formed without any appreciable change to hardware or monitor software.

Using this approach, the only requirement a system must meet to be eligible for membership in the network is the following: the time-sharing monitor must allow user programs to communicate with *two* terminals. If this requirement is uniformly fulfilled, then the network can be implemented without change to the monitor at any installation, by the simple expedient of letting each computer in the network look upon all the others as though they were its own remote-user terminals.

Figuratively speaking, we may think of the computer-to-computer link in such a network as being the result of removing a user terminal from its cable on computer X, removing a user terminal from its cable on computer Y, and splicing the two cable-ends together.

* It may be pointed out that the type of network described above is not the only possible type. One might alternately consider a network in which programs are shipped from one computer to another. Such a program-shipping network could be used for load-sharing: When the queue of programs waiting to run on a given computer becomes too long, certain of the programs are shipped off to another computer where quicker service is available. The establishment of a program-shipping network seems to us to be an exceedingly difficult undertaking and will *not* be considered in the present paper. Instead, we restrict ourselves entirely to the type of network in which all programs run exclusively on their home computer, that is, on the computer they were programmed for.

Such a network operates as follows. The user dials up his home computer, C_H , from a console. He logs in normally by transmitting characters from his console to the monitor M_H . He sets up his user program P_H and starts this program. P_H , through the second channel available to it, logs in at the remote computer C_R , by transmitting the correct sequence of symbols to the remote monitor M_R . (Note that it is the user's program P_H , not the monitor M_H , which has the responsibility for doing this.) The remote monitor M_R takes the proper actions and communicates with P_H that the actions have been taken. P_H accepts these messages. P_H then communicates with M_R to set up the desired program P_R at the remote computer; it then runs P_R and transmits and accepts data from it, until it is done. P_H then logs out by communicating with M_R . P_H continues on its own until done. The user logs out by communicating with M_H .

Note that neither M_H nor M_R was required to behave in an unusual fashion. The monitors did what they always do. The only requirement, as stated earlier, was that the user program P_H be allowed to communicate with two terminals, its own user terminal and the remote computer. Most present-day monitors provide for such a capability.

There are a number of problems with this elementary approach. First, while it is true that most present-day monitors allow the user program to communicate with two terminals, they typically allow this communication to occur only at very low data-rates. This is because the multiple remote-access lines which monitors are designed to service are teletypewriter lines, operating at teletypewriter data-rates, i.e., at rates on the order of 100 bits per second. In the following section we discuss alternative approaches to circumvent this difficulty.

Second, the elementary approach leads to a network which in many circumstances is quite inconvenient. One reason is that the responsibility for making the network operate rests on the calling program P_H . This program must handle the communication with the remote monitor, possible code conversion, error checks, etc., without any assistance from the monitor. Another reason is that there is no way of "reaching" remote programs other than those which take all inputs from, and give all outputs to, a user terminal. Ways of dealing with this second class of problems will be discussed below under the headings of Message Protocol, Auxiliary Software, and The Problem of Displays.

Achieving Higher Data-Rates

We have seen how an elementary network could be achieved without changing existing hardware or monitor software. This elementary network operates at teletype data-rates, i.e., at rates on the order of 100 bits per second. We consider next how to achieve higher data-rates while still preserving, as much as possible, the hardware and software of existing time-sharing systems. Two possibilities are open to us:

1. The limitation on the rate at which bits are transmitted to and from teletypewriter stations is imposed by the speed of the teletypewriters, not by the capabilities of the hardware interfacing the remote lines to the computer. Trivial modifications to this communications interface will frequently allow the bit-rate to be increased considerably over the teletype rate. An improvement by a factor of 10, leading to a rate of approximately 1000 bits per second, should not strain any existing hardware. Such a change in the hardware does not require any change in the monitor software, provided of course, that we do not exceed the monitor's capacity for accepting characters.

2. The channels considered up to now have been command-plus-data channels which carry commands (from user to monitor) as well as data (from user to user-program and vice versa). Since commands come at unpredictable times in the data-stream, each character transferred over a command-plus-data channel must be analyzed by the monitor. Not so for the case of data-only channels, such as those which transfer information between a user program and a disc file or a magnetic tape. The data-flow over a data-only channel contains no information addressed to the monitor itself; hence the information need not be analyzed by the monitor and can be transferred at higher rates.

This fact suggests that data-rates higher than those discussed up to now could be achieved by using data-only channels as computer-to-computer links; such channels are already available in existing time-sharing systems, and could be modified for networking applications.

However, data-only links would not be sufficient for a network, since it is necessary to transmit commands to the monitors (in order to log in, for example); primary data-only links would have to be supplemented by secondary command-plus-data channels.

Thus, a possible alternative technique for achieving increased data-rates without greatly increasing the burden on the monitor would be to use high-rate data-only links, supplementing these by low-rate command-plus-data channels over which communication to the remote monitor could take place.

Message Protocol

We have seen how an elementary network, operating at low data-rates, could be implemented with virtually no changes to existing hardware or software, and how somewhat higher data-rates could also be achieved without radical redesign. However, the networks created by the principles discussed so far have great shortcomings, particularly since they require the user program initiating the call to the remote computer to do all the work associated with carrying out the transmissions. This can be a very great inconvenience; it would be much more desirable to have the monitor take over some of the chores.

The first step in that direction is the establishment of a message protocol, by which we mean a uniform agreed-upon manner of exchanging messages between two computers in the network.

The primary reasons for considering the establishment of a message protocol are the following:

1. By formatting transmissions into messages, and including a check-sum with each message, transmission errors can frequently be detected. If detected, the messages can automatically be retransmitted in accordance with the protocol.
2. If an existing command-plus-data link is used, certain characters or strings of characters are normally given special consideration by the monitor. Hence, binary data cannot efficiently be sent over such a link without some extra provision which specifies that a certain block is to be regarded as binary information. The protocol provides for such an extra provision and thereby allows binary information to be transmitted efficiently.

As will be seen below, work is proceeding on an experimental network between the TX-2 computer at Lincoln Laboratory and the Q-32 computer at

System Development Corporation. The protocol to be used in this network is given in the Appendix. This protocol will be handled as an extension of the two monitors, which will automatically take care of error-checks, retransmissions, and acknowledgments.

While the implementation of a message protocol is a great convenience to the user, a certain cautionary remark may be in order. If a protocol is adhered to between two computers in the network, say A and B, it is not absolutely necessary that the same protocol be established between C and D, nor even between A and D. Since the motivation for the network is to overcome the problems of computer incompatibility without enforcing standardization, it would not do to require adherence to a standard protocol as a prerequisite of membership in the network. Instead, the network should be designed for maximum flexibility. If a protocol which is good enough to be put forward as a standard is designed, adherence to this standard should be encouraged but not required.

Auxiliary Software

Let us assume that we have a command-plus-data channel over which computers X and Y communicate, and that a message protocol has been arranged. Consider now the auxiliary software necessary to make use of the networking capability.

Let us say that X is the home computer (the computer on which the user is logged in) and that Y is the remote computer. There are two types of programs on Y that are of interest to the user of X:

1. "Total program packages," that is to say those programs all of whose inputs are typed in by the user and all of whose outputs are typed out to the user. An example of such a package is an on-line engineering-calculation program.
2. Subroutines, that is to say those programs which are called by other programs, whose arguments are transmitted at time of call, and which return control to the calling program together with the result of the calculation.

Consider now the auxiliary software that needs to be provided *at the remote computer Y* in order for the two types of programs to be usable remotely from X.

- To use a total program package remotely, no additional software is necessary at Y.
- To use a subroutine remotely, a *user program* running on Y must be provided. This user program is an interface between the link and the desired subroutine. The user program accepts type-in from the link, calls and runs the desired subroutine, and types out the answer to the link. A separate user program could be provided for each subroutine of interest; or else a common program could be written which is given the arguments and the name of the desired subroutine.

Consider next the auxiliary software that needs to be provided *at the home computer X*. For each remote program that is to be used, code needs to be written for calling up the remote computer, logging in, calling up the program, transferring data, and logging out. This code may as well be public, so as to be available to all users of X. Thus, if the programs ABLE and BAKER running on computer Y are to be used remotely from X, programs PSEUDOABLE and PSEUDOBAKER will be provided on X. These PSEUDO-programs will be called by the user-programs of X in the same manner as ordinary public programs at X. Since ABLE and BAKER run on the same computer, Y, PSEUDOABLE and PSEUDOBAKER will have a great deal of overlap (the dial-up and log-in routines for Y, for example). Hence PSEUDOABLE and PSEUDOBAKER will probably be written in such a way as to employ a common subroutine which handles all communications with Y. Given this routine, PSEUDOABLE and PSEUDOBAKER are trivial to write.

The Problem of Displays

There are certain special problems associated with the remote use of display programs. The first of these deals with the fact that display programs fit into neither of the two program categories discussed above. Display programs are not total program packages, since they do not type out their results; nor are they subroutines in the sense defined, since they do not return their results to the calling program, but instead send these results to a display-generator.

In order for an *existing* display program to be used remotely, therefore, it is not sufficient to introduce auxiliary software as user programs. A change to the monitor is required.

What is required is (1) that the monitor recognize the situation in which the user who is calling up the display program is not an ordinary user but rather a remote computer and (2) that it take special action in this case regarding the display information. The special action required is to ship the display information out over the link rather than to the local display. If such a monitor modification is made, existing user programs involving displays can then be called from the remote computer, and the display information will be sent to the home computer.

The second problem arises from the fact that there is no agreed-upon language for transmitting displays. The handling of displays in a time-sharing system is usually built right into the time-sharing monitor. Unfortunately, every monitor handles displays differently, with the result that each computer in the network must be programmed to understand the display languages of the others. This can certainly be done, but it is inconvenient.*

It might be pointed out that if we solve the two problems stated above in a satisfactory manner, so that display programs can be used remotely in a network of time-shared computers, we will ipso facto have solved also another problem of current interest, namely the problem of small satellite computers used as remote display consoles. In fact, there is no reason why satellite computers and network computers need be treated differently. A satellite computer also must log in, run a program, and have display information transmitted back down the communication channel.

HARDWARE CONSIDERATIONS

There exists a multitude of hardware problems that must be considered when a computer network is planned. Decisions must be made regarding type and speed of communication channels, character size, serial versus parallel transmission, synchronous versus asynchronous transmission, full-duplex versus half-duplex, etc. Since these problems have been dis-

* As in the case of message protocol, adherence to a proposed standard should be encouraged but not made mandatory. In any event, the problem at the moment is not lack of adherence to a standard, but rather lack of any proposed standard.

cussed in detail elsewhere,¹ and will also be considered by other contributors to the present conference, no further review will be undertaken here.

One comment may be in order. Automatic calling units (devices which allow computers to place calls over common-carrier dial-up networks) are becoming available. These devices allow a computer to place a call to a remote computer at the time the connection becomes necessary, eliminating the need for permanent connections. In fact, if the remote computer is to perform a lengthy computation, it would be feasible to start the remote program, hang up, and have the remote program call back when it gets the answer.

NETWORK EXPERIMENTS

Work is proceeding on the implementation of an experimental network involving the APEX time-sharing system² running on the TX-2 computer at MIT Lincoln Laboratory in Lexington, Massachusetts, and the time-sharing system running on the Q-32/PDP-1 computer complex at System Development Corporation in Santa Monica, California.³ Initially, a 4KC four-wire dial-up system will be used with 1200-bit-per-second asynchronous modems. The message protocol given in the Appendix will be used. In addition, a special display language is being developed, and suitable monitor changes are planned, so that display programs can also be used remotely.

As soon as possible, a series of demonstrations and experiments will be performed using the experimental network. The experience gained will be reported at the conference. If the outcome of the experiments supports the validity of the concepts, it is hoped that other time-sharing installations will join the experimental network.

APPENDIX

MESSAGE PROTOCOL FOR TX-2/Q-32 LINK

This Appendix describes the message protocol for use with the link between the Q-32 at System Development Corporation in Santa Monica, California, and the TX-2 at Lincoln Laboratory in Lexington, Massachusetts.

Each character consists of eight data-bits, sent least significant bit first, preceded by a zero start bit and followed by a one stop bit. When not transmit-

Table 1. Special Characters for Message Protocol

<i>Octal</i>	<i>ASCII</i>	<i>Meaning</i>
HEADER		
201	SOH	characters for monitor
202	STX	characters for user
221	DC1	data for monitor
232	SS	data for user
END OF MESSAGE		
203	ETX	end of message
ACKNOWLEDGMENT		
225	NACK	message in error, repeat
234	FS	message OK, but wait
206	ACK	message OK, send next message
QUERY		
230	CNCL	resend last acknowledgment
SYNCHRONIZATION		
226	SYNC	ignore
SPECIAL FUNCTIONS		
220	DLE	help/break
233	ESC	panic.

ting a character, the link transmits a one continuously.

All information transmitted is sent in the form of messages consisting of a header character, body, end-of-message character, and a checksum. All messages are acknowledged.

There are four types of messages. Each has a unique header character that determines both the destination of the message (user or monitor) and the mode of the message (character string or binary data). The specific characters used are listed in Table 1.

The body of the message has a maximum length of 119 characters if the message is a character string and 118 characters if the message is binary data. If the message consists of binary data, the first character of the body is a count character equal to the total number of characters in the body including the count character. Two through 118 are legal values.

The body of the message is followed by an end-of-message character. This is immediately followed by a checksum—the 8-bit ring-sum of the header character and all the characters in the body.

A message is acknowledged by sending one of three characters. One indicates an error, requesting retransmission. The second indicates that the message was received correctly, but requests that the transmitter wait before sending the next message, as the receiver buffers are full. The third type of ac-

knowledge indicates that the last message was received correctly and/or that the receiver is ready for the next message.

There are four other special characters. The first—query—requests the receiver to resend the last acknowledgment character that it sent, or to indicate an error if it is currently receiving a message or has received garbage since the last correct message.

The second—sync—will be used by another (synchronous) link attached to TX-2. As far as the SDC/TX-2 link is concerned, sync characters are ignored.

The other two—help and panic—are both treated as a break at SDC or help request at Lincoln. Eventually panic will be used for a higher-level interrupt at Lincoln.

As described herein, the system is capable of transmitting and receiving messages simultaneously. To speed up text transmission, acknowledgments and queries may be interjected in the middle of character strings (not binary data) as the receiver will always be looking for special characters when in the character mode. Interjected characters are not included in the checksum.

Since SDC desires not to transmit and receive simultaneously, programs using this system should be arranged to alternate messages or groups of messages.

To avoid "hung" conditions, the transmitter is responsible for getting the message through and acknowledged. If the transmitter does not receive an expected acknowledgment within one second, a query is sent to see if the message or acknowledgment was lost. Similarly, if a ready condition (ACK) is not received within 30 seconds of a "wait" (FS), a query is sent to determine if the ready was lost.

All special characters have the high order bit set to one. This leaves all 128 characters whose high

order bit is a zero available to transmit the full 7-bit ASCII code in the character mode. Care must be exercised by programs using this system, since it is possible to send characters that could not originate from a teletype. When in the character mode, characters whose high order bit is a one are ignored if they are not special characters.

ACKNOWLEDGMENTS

The authors wish to thank the many individuals who have generously taken the time to discuss computer networks with them: at System Development Corporation, C. Fox, D. Kemper, L. Gallenson, J. Schwartz, R. von Buelow, C. Weissman; at MIT Project MAC, S. Dunton, D. Edwards, R. Stotz; at Lincoln Laboratory, J. Forgie, K. Konkle, J. Mitchell, J. Raffel; at Computer Corporation of America, W. Mann, H. Murray. In addition, the people at System Development Corporation, Lincoln Laboratory, and Computer Corporation of America are participating in setting up the experimental network, and their cooperation is very much appreciated.

REFERENCES

1. T. Marill, "A Cooperative Network of Time-Sharing Computers: Preliminary Study," Technical Report No. 11, Computer Corporation of America, Cambridge, Mass. (1966).
2. J. W. Forgie, "A Time- and Memory-Sharing Executive Program for Quick-Response On-Line Applications," *Proc. Fall Joint Computer Conf.*, vol. 27, part 1, Spartan Books, Washington, D.C., 1965, pp. 599-609.
3. J. I. Schwartz, E. G. Coffman, and C. Weissman, "A General-Purpose Time-Sharing System," Document SP-1499, System Development Corporation, Santa Monica, Calif. (1964).

THE LINCOLN RECKONER: AN OPERATION-ORIENTED, ON-LINE FACILITY WITH DISTRIBUTED CONTROL

Arthur N. Stowe, Raymond A. Wiesen, Douwe B.
Yntema, and James W. Forgie

Lincoln Laboratory, Massachusetts Institute of
Technology
Lexington, Massachusetts*

GENERAL DESCRIPTION OF THE RECKONER

The Lincoln Reckoner is a time-shared system for on-line use in scientific and engineering research. It looks forward to the day when a computational service can be put into the office of every serious research worker, and it was designed as an experiment to find out what features of such a service will have an important effect on the amount of work the user gets done.

The present system does not by any means offer all of the services that a computer might provide to scientists and engineers. Instead, it offers a library of routines that concentrate on one particular application, numerical computations on arrays of data. It is intended for use in feeling one's way through the reduction of data from a laboratory experiment, or in trying out theoretical computations of moderate size.

Using Routines from the Public Library

The Reckoner is primarily a facility for making use of routines, not for writing them. To run a rou-

* Operated with support from the U.S. Air Force.

time, all the user has to do is type the name of the routine, the name or names of the operands on which he wants it to operate, and the name or names under which he wants the results to be filed away for future reference.

Suppose for example that two arrays, X and Y , have already been loaded into the system: X is an $m \times n$ array and Y is $m \times 1$. The user would like to find the coefficients β_j that minimize the expression

$$\sum_i (y_i - \sum_j x_{ij} \beta_j)^2$$

Let us assume he knows some matrix algebra: he realizes that the desired coefficients are the components of the column-vector β that may be found by solving the matrix equation

$$X' \times X \times \beta = X' \times Y$$

in which X and Y are the given arrays and X' is the transpose of X . To find β he might then type the four lines shown in Example 1.

Example 1:

```
TRANSPOSE X XPRIME
MATMUL XPRIME X L
MATMUL XPRIME Y R
SOLVE L R BETA
```

The first line says that the array named X is to be transposed and the resulting array is to be called XPRIME. TRANSPOSE is the name of the routine that transposes a matrix (i.e., a two-dimensional array), X is the name of the array on which the routine is to operate, and XPRIME is the name under which the user wants the result to be filed away. When he has typed this line and has pressed the carriage-return key on his typewriter, the system performs the operation without further ado. He need not know where TRANSPOSE and X are stored, and he need not concern himself with the dimensions of the operand X or of the result XPRIME.

The second line says that the matrix XPRIME is to be post-multiplied by the matrix X and the product is to be called L. MATMUL is the name of the matrix-multiplication routine, there are two operands, named XPRIME and X, and L is the name the user wants to give to the result.

The third line is similar, and the fourth says that the matrix equation

$$L \times \beta = R$$

is to be solved for β . L and R are the names of the operands, SOLVE is the name of the routine, and BETA is the name for the result.

At this point the user may want to see how well each of the approximations $\sum_j x_{ij} \beta_j$ agrees with the corresponding y_i . He might therefore type

```
MATMUL X BETA XBETA
PLOT XBETA Y
```

The first line multiplies the $m \times n$ matrix X by the $n \times 1$ matrix BETA to produce the $m \times 1$ matrix XBETA, and the second produces on a cathode-ray tube near the user's typewriter a graph of the elements of XBETA plotted against the corresponding elements of Y. The routine chooses the scales for the two axes, puts tick marks on the scales at intervals chosen to conform to custom and readability, and decides whether to label every tick mark or every fifth one. There are other routines, with additional parameters, that allow the user to specify the ranges of one or both of the axes.

Building and Using a "Process"

As the user works on a problem, he often finds himself using over and over a series of operations that he has come to regard as a single unit. When he

does, he will probably want to define a "process" that will allow him to run off the whole sequence with a single statement. For example, he may come to think of the four lines shown in Example 1 as one unit, a least-squares fit. To make the four lines into a process he could proceed as shown in Example 2.

Example 2:

```
BUILD LSF
GO ON
TRANSPOSE X XPRIME
MATMUL XPRIME X L
MATMUL XPRIME Y R
SOLVE L R BETA
| FINIS
OK
```

On the first line, BUILD is the name of the routine that constructs a new process, and LSF (for "least-squares fit") is the name the user wants to give the process. The system responds GO ON to show it is ready to accept the definition of a process, and he types the four lines that constitute the definition. Then he types |FINIS to show he is through, and the system responds OK. Henceforth, whenever he types LSF followed by a carriage return, the system will perform the four statements exactly as though a rapid typist had typed them very quickly. Processes can call upon processes: the statement LSF can appear in the definition of another process, which can be called by yet another process, and so on.

Sometimes it is convenient to let a process have operands and results so that it can be used like one of the primitive routines in the library. If the user had chosen to make LSF a process of this kind, he could have proceeded as in Example 3.

Example 3:

```
BUILD LSF
GO ON
p A B C
TRANSPOSE A .APRIME
MATMUL .APRIME A .L
MATMUL .APRIME B .R
SOLVE .L .R C
|FINIS
OK
```

The line beginning with the circled *p* is the parameter line: it shows which of the names that appear in the process are to be treated as variables that the user will specify when he asks for the proc-

ess to be performed. For example, if he types the line

LSF X Y BETA

the process will be run off just as though the four statements were typed by a rapid typist who was instructed to substitute X where A appears, Y wherever B appears, and BETA where C appears.

The curious names that begin with periods are names for intermediate results that the user does not want preserved after the execution of the process is complete. The system automatically adds to each of these names a prefix composed of a period and the name of the process itself. For example, wherever the user has written .R, the fictitious typist writes .LSF .R, so that .R becomes in effect a local name peculiar to this process; thus if the statement

LSF X Y BETA

appears in another process in which the name .R is used, the two results called .R will not be confused. When the execution of the process is complete, the system automatically makes the name .LSF .R undefined so that it will not clutter up the user's records.

The versatility of processes is greatly increased by two features that have not been illustrated by these examples. First, processes may include unconditional and conditional jumps. For example, when the line

5.5

appears in a process, it means "jump to Line 5.5 of this process"; and the line

5.5 ALPHA > Z

means "jump to Line 5.5 of this process if the current value of the scalar named ALPHA is greater than the current value of the scalar named Z." (The lines of a process all have numbers, though the user need not bother to type them if he wants the lines numbered simply 1, 2, 3, . . .) Second, there is a symbol, Φ , which means "now exit from this process." This symbol did not appear in the examples because a process is always understood to have an exit at the end.

Since the user is able to construct processes that contain conditional jumps and behave like library routines, the public library of basic routines does not have to be very large. The present library, which is specialized for numerical calculations on arrays of real numbers, is described in the Appendix. It contains only 64 operations—some large, many trivial. While it is easy to think of other operations that

might well be included, we have been pleased and a little surprised to find how adequate this small library is.

Further Details

The Reckoner has some other features that should be mentioned because they greatly affect the appearance of the system to the user. It is easiest just to list them:

1. There is a buffer area in which the system stacks inputs from the user's keyboard. After typing a carriage return, which is the signal to execute a statement, he may begin a new statement immediately without waiting for an indication that the system has finished executing the old one. Stacking inputs in this fashion is a cheap way to increase the apparent speed with which the system responds: the system can be working on one statement while the user is typing the next. Indeed he never needs to wait for the system except when he types a command that calls for an output, and when he goes into or out of the process-builder. (In theory he would have to wait if he typed fast enough to fill the buffer, which is set to hold about 700 characters; but no one has ever filled it so far as we know.) There is of course an unpleasant moment of confusion if he is typing a statement when the system begins to type an error-message about some previous statement, but the occasional unpleasantness is a small price to pay for the apparent increase in speed of response.

2. The user may edit the definition of a process: he may make changes before he has typed the FINIS that shows the definition is complete, and he may reopen the process later, perhaps after he has tried it, and make changes then. Lines are replaced and deleted by number, and are inserted by giving them intermediate line-numbers. Line-numbers may have up to four decimal digits before and four after the decimal point.

3. A name may be 50 characters long, and may consist of any combination of digits, periods, and roman capital letters, provided it contains at least one letter. However, it is unwise to choose a name beginning with a period or a digit; names that begin with periods can be confused with names of intermediate results, and names that begin with digits are sometimes used by the system for its own purposes.

4. There is an operation (i.e., a public routine) by which the user may specify that one name shall

have the same meaning as another. For example, the basic public name of the matrix-multiplication routine is *5MMUL*. If the user does not like that name—and presumably he will not—he may create a synonym like *MATMUL*, the name used in the examples. He may also create synonyms for the names of his own processes and arrays of data. In particular, he may create one- or two-letter synonyms for names he uses often.

5. When a name is given a new meaning—either by making it a synonym to another name, or by using it as the name for the result of an operation—it is detached from its old meaning, if any, and attached to the new meaning; if it did not have synonyms, the thing to which it referred is forgotten. The system does all this without any comment or warning. This arrangement appears to be proper and convenient from the user's point of view; but it does imply that when he wants to protect a valuable file of data, he should give it a long, complicated name that he is not likely to reuse inadvertently.

6. A process may be interrupted by the interrupt button, by a special statement in the process itself, or by an error message. The process is put into a suspended state—and so is the process, if any, that called for it, and the process which called for that one, and so on. Control is returned to the keyboard, and the user may print the results obtained so far, plot them, or do subsidiary calculations on them. In doing so he uses library routines and existing processes just as he usually would; he is even free to define and use new processes. He uses names like *.LSF.R*, which was discussed in connection with Example 3, when he wants to refer to intermediate results of a process that is in suspension. If he wishes he may tell the system to continue the complex of suspended processes, starting where it left off.

There are several other details of more interest to the professional programmer than to the typical user: (1) The routines in the present library do all calculations in single-precision, floating-point arithmetic (9-bit exponent; 27-bit fraction, including sign) and simply take the nearest integer when an integer is needed—e.g., for the number of rows in an array. (2) At present all arrays are n -dimensional rectangular arrays of single-precision, floating-point numbers; the number of elements in an array may not exceed $8190n$. (3) The routines in the public library are binary, reentrant, closed subroutines that occupy fixed addresses; processes run in-

terpretively, as will be explained below. (4) Intermediate results of the sort illustrated in Example 3 are handled in such a way that a process can be recursive: if the process is called *PROC*, and it is already in use on another level, then the prefix added to names of intermediate results is *.1 .PROC* instead of *.PROC*; if it is already in use on two levels, then the prefix is *.2 .PROC*; and so forth.

Three Case Histories

Our general impression is that the Reckoner is a good, workaday facility for doing calculations with a minimum of fuss about clerical details. We have chosen three case histories to illustrate how the system works out in practice. The Reckoner is sometimes used to attack sizable problems that require many hours of work, but for the sake of brevity, we have chosen short cases.

Case A. Senior member of the Laboratory's staff; poor touch-typist; had written programs as pencil and paper exercises but never run them; using the Reckoner for the first time, spending an hour trying it out with the help of an experienced user.

Spent about 10 minutes typing small arrays into the system and doing various operations on them more or less at random. Then turned to a problem that had arisen the previous day in his work: Iteratively compute the vector S from the equation

$$\frac{C \times S_p + V}{6} = S_n$$

where C is a square matrix, V is a column-vector with each element equal to 2, S_p is the value of S from the previous iteration, and S_n is the new value. Typed a 5×5 matrix C and an initial vector S into the system, and with occasional coaching from the experienced user, did one round of the iteration and typed out the elements of S . Applied the process repeatedly until S stabilized.

Did not believe the results, and spent about five minutes with pencil and paper thinking about the physics of the problem. Tried another initial value of S , got the same results, and then saw they were reasonable. Asked if he wanted to save his files, said, "No, I have my answer."

Time: 55 minutes, including some discussion of the purpose of the Reckoner.

Case B. About two hours previous experience with the Reckoner; fair-to-middling typist; had written

and used two small programs in machine language; no acquaintance with languages like ALGOL or FORTRAN.

Communication-satellite problem that had arisen in a study group in which he had participated. Considerable advance preparation in his office (amount of time unknown) reading documents about the Reckoner, doing algebra on the blackboard, and writing out two processes on paper.

At the console, began by typing latitudes and longitudes of six cities around the globe. Assumed four communication satellites in polar orbits: for simplicity, circular orbits all of the same altitude. Typed in initial latitudes and longitudes of the satellites.

Defined a process that computed which satellites were visible from which cities. Did not believe the answers, and looked back through the results of intermediate calculations. Found he had gotten confused about whether sine or cosine of 60° was $\frac{1}{2}$. Corrected the error and redid the computation. Then printed out a 6×6 table with a row and a column for every city, each entry being the number of satellites through which the two cities were able to communicate.

Defined another process, which advanced time—i.e., rotated the earth by a certain number of minutes and moved each satellite around in its orbit by the same number of minutes. The process then computed which satellites could be seen from which cities and printed the 6×6 table.

Using this process, ran the simulated communication-system through most of a day's operation, drawing conclusions about the way the initial placement of the satellites had affected the continuity of communication.

Time: 1 hour and 45 minutes in a continuous session, plus an unknown amount of advance preparation. The time is particularly impressive because the Reckoner did not yet have routines for sines and cosines; the user was continually calculating his way around their absence.

Case C. Very experienced user, one of the designers of the system; facile typist.

Was demonstrating the Reckoner to a prospective user who had spent several hours the previous day with a desk calculator finding how well

$$\frac{\cos\left(\frac{\pi}{2} \cos \theta\right)}{\sin \theta}$$

is approximated by $\sin \theta$ in the interval 0 to $\pi/2$.

To show how the computation would be done on the Reckoner, typed a six-statement process, which when given an array of numbers, θ , created a new array, each of whose elements was derived from the corresponding element of θ according to the expression

$$\sin \theta - \frac{\cos\left(\frac{\pi}{2} \cos \theta\right)}{\sin \theta}$$

(Note: This is a case where an algebraic language would clearly have been helpful.)

Starting with the operation that creates a one-row array of the integers from 1 to N , then multiplying each integer by $\pi/2N$, created an array THETA that contained a couple of hundred numbers equally spaced between 0 and $\pi/2$ (0 excluded). Applied the process to this array and plotted the results against the elements of THETA. Saw that the magnitude of the expression given above is maximum in the vicinity of $\theta = 0.6$; so asked for an expanded graph of the function in that region and read off the value of the maximum as 0.0850

Time: Under 10 minutes, including time to state the problem.

OPERATION-ORIENTED SYSTEMS

The Reckoner may be said to belong to a class of systems that includes the OPS system¹ at Project MAC, the MAP system,² which is also at MAC, the AMTRAN system³ at the Marshall Space Flight Center, the system on which Culler is working at Santa Barbara,⁴ and the earlier Culler-Fried system,⁵ which influenced the design of the Reckoner in many ways.

All of these systems have the following three features:

1. *Automatic application of routines.* Almost all of the clerical work needed to perform an operation—i.e., to apply a public routine—is done automatically. The system takes care of the location of the data, the dimensions of arrays, and so forth. Ideally, all the user has to do is somehow indicate the operation he wants to apply, the data to which he wants it applied, and—perhaps—the way in which he will identify the results when he wants to use them again.

We say “perhaps” because some of the systems have a sort of accumulator from which the typical

operation takes one of its operands and in which it deposits the result; thus a result that is to be used only in the next operation need not be stored away. And we say "almost all the clerical work" because the ideal can be approached but never achieved entirely.

2. *Automatic retention of results in such a form that they can be used as operands for other routines.* The results of operations are stored in such a fashion that they can be used later as inputs to other operations—including operations that the user did not have in mind when the results were obtained. He need only specify the name or the button by which he wants to identify a result; the system remembers where the result has been stored and automatically records the descriptive information that will be needed if the result is to be used later as an operand—e.g., the dimensions are recorded if the result is an array of numbers.

3. *Facilities for concatenation of routines.* The user can define a sequence of operations and then use the sequence as he would use one of the primitive routines in the library. The new operation can be used as part of another sequence, and so on.

We propose that the class of systems defined by those three characteristics be called "operation-oriented." To the user, a process or a primitive routine is just an operation that he may perform on files of data to produce graphs, printed tables, or new files of data. In a different sense the word "operation" is appropriate from the designer's viewpoint, too. These systems are oriented not toward the preparation of routines, but toward the operation of them. There may of course be provisions for writing a new routine; but whether there are or not, convenience in the execution of existing routines is essential.

It should be emphasized that operation-oriented systems need not look alike to the user. A system whose library was specialized for the manipulation of text would look very different from the present Reckoner.

Appropriateness

The users for whom we are designing are scientific and engineering research workers who may not know much about programming, but who could benefit from computational assistance, especially in

feeling through moderate amounts of data or through relatively small theoretical calculations.

Operation-oriented systems seem appropriate for that kind of user. The present system indicates it is feasible to provide routines that can be combined to perform most of the operations that the user wants; so it is efficient for the system to emphasize the convenient use of routines from a library. A user who is feeling his way through a problem often has no clear idea of what he will do next; so it is efficient to retain results in such a form that they can be used as inputs to operations that were not foreseen when the results were obtained. And as the user gets familiar with a problem, he generally thinks of the calculations in larger and larger units; so it is efficient to let him build processes that conform to those units.

The advantages of an operation-oriented system extend beyond the use of library routines. No matter how complete the library may be, the user, or his programming assistant, will sometimes have to write new routines to perform operations that cannot be satisfactorily contrived from the operations in the library. Some operation-oriented systems therefore provide access to procedure-oriented languages like FORTRAN and MAD, and a small procedure language is being added to the Reckoner. To be most useful, a procedure language must allow one to write routines that can be used like the routines in the library. A new routine will generally be used a number of times with different parameters or with different sets of data; so convenience in applying it is important. Furthermore, the programmer has less work to do if he can write a routine that accepts the results of library routines as inputs and stores its results in such a form that library routines can employ them. He can concentrate on the crucial aspects of the new operation, and can rely on the library for preparation of the inputs, inspection of the results, and so on.

Problem-oriented languages are also very useful to the class of people for whom we are designing. But if "problem-oriented" is understood in its strict sense—oriented toward the problems of a particular field like statistics or stress analysis—then these languages have limitations. A research scientist or engineer will inevitably push against the limits of any problem language that can be provided for him. After all, his job is partly to define new problems. He will often want to step outside of the language provided for him, apply his own processes, and then

go back. Thus problem languages, like procedure languages, will be more useful if they are embedded in an operation-oriented system.

Indeed, it seems that the three characteristics we have listed would be desirable in any work that a researcher might want to do on-line—editing FORTRAN programs, making drawings, writing memoranda, and all the rest. We are almost prepared to argue that any system for on-line use in a research laboratory should be operation-oriented.

THE RECKONER FROM THE PROGRAMMER'S POINT OF VIEW

This section will describe how the Reckoner is put together and will comment on some decisions that were made in designing it.

Services Provided by the Time-Sharing System

The Reckoner is time-shared and runs on TX-2, the Laboratory's experimental computer. The TX-2 time-sharing system, which is called APEX, provides the Reckoner with three important services: the input buffer, the directory of names, and the stack of "maps." We shall just summarize them; they have been described elsewhere.⁶

The input stacker takes characters from the keyboard and puts them into a buffer area. The executive notifies the user's program when each statement is complete—i.e., when the user has sent a carriage return—but it continues to stack further statements, up to the capacity of the buffer, whether the user's program has processed the previous ones or not.

The executive maintains a private directory for each user, and also a public directory of library routines and the like. In either case, the directory is a ring-structured dictionary of names and definitions. If the name refers to a single number, then the definition consists simply of the number, but if the name refers to a file (either a routine or a file of data) then the definition includes an indication of where the file may be found and some of the descriptive information about it. The user's directory can contain a name defined by a pointer to a definition in the public directory, and there can even be names with no definitions (an important provision in a system of this type). The user's programs are not allowed to write in the directories, but they can manipulate the private directory and inspect both directories through calls to the executive.

The executive also maintains the user's stack of "maps." APEX provides each user with an "apparent computer" whose core memory is divided into 16 segments. A segment can contain only one file, and a "map" is a list specifying the name of the file, if any, that goes in each segment. A program may of course get access to a subroutine by asking the executive to put the subroutine in a given segment on the current map. But to make it easy for one routine to use another when their memory-address requirements conflict, the calling routine may direct APEX to put the new routine on a fresh map and transfer control to the new routine, remembering the specification of the old map. This process is called "going up" to a new map, because the new map is regarded as stacked on top of the old one. The reverse process, returning to the calling routine on the next lower map, is called "peeling back." Common to all maps is a special file, called the connector, which is used for passing information between routines on different maps. By convention, the first register in the connector contains a pointer to the first register that is not currently in use.

The input buffer, the directory, and the stack of maps were built into the APEX executive partly for reasons that depend on the nature of TX-2, and partly because APEX was intended to support operation-oriented systems. These three services must be provided in some form, but they do not necessarily have to be built into the time-sharing executive.

The Basic Translator

The execution of a library routine or a process that the user has called from the keyboard begins with the basic translator. It takes the statement out of the buffer area, cleans up deleted characters and so on, and then divides the statement into its component terms, which are usually separated by spaces. Terms that are text—i.e., material between the characters that are used as quotation marks—are set aside, and terms that satisfy the rules for numbers are converted to numbers in binary form. Any term that is neither a quote nor a number is assumed to be a name, and the basic translator asks APEX to look it up. APEX looks first in the user's private directory and then in the public directory, and if the name does not appear in either directory, inserts it in the private directory as a name that is not yet defined. In either case APEX replies with a

pointer to the location of the name in one of the directories.

The translator types an error-message if it is unable to interpret one of the terms as a quote or a number and is unable to get the directory to accept the term as a name. It also gives an error-message if the first term in the statement is not the name of a binary, executable routine.

If there are no errors, the translator puts into the connector a calling sequence that is a term-for-term translation of the statement the user typed: quotes are given literally, numbers appear as actual binary numbers, and names are represented by pointers to the directory. Finally, the translator issues a "go up" call to APEX, giving as a parameter of the call the pointer to the first name in the statement, the name of the routine that is to be executed.

The subroutines for number conversion and for cleaning up the statement are fairly sizeable, but otherwise the basic translator is rather simple. It can be simple because its only responsibility is to construct a calling sequence and then go up. It balks at going up to anything except an executable routine, but aside from that, it does not know anything about the nature of the routine that it is calling: it does not know which terms the routine will use as inputs, which it will use as names of outputs, or how many terms are expected; it does not even know whether it is calling a process or a library routine. Letting the basic translator work in such ignorance is an example of a policy that we shall call distributed control. We shall come back to that later.

Library Routines

When a library routine has been entered by going up to it, it gets its instructions from the calling sequence in the connector. If it is to operate on files of data, it takes from the calling sequence pointers to the names of the files and asks APEX to set them up on the current map. It computes from the data descriptions in those files the amount of space it will need for working storage and for results, and it asks APEX to create on the current map new files of the appropriate sizes.

The routine then proceeds to do the computations that are its main task, and inserts into the file (or files) of results whatever descriptive information is conventional for files of that type. It then takes from the connector a pointer to the name that the results are to have, and passes the pointer to APEX with a

request that the file be given this name. (The naming of the results is done last so that if the name of an input is to be the same as the name for a result, the name will not change meaning until the routine is sure that it can finish successfully.) And finally, the routine peels back to whatever called it.

Since the basic translator does not know the requirements of the routines that it calls, a library routine must check the acceptability of the terms that appear in the calling sequence, and the acceptability of the data to which they refer. It must see that the sequence contains the right number of terms, that the term for the result is a directory pointer and not a number, that the dimensions of arrays on which it is to operate are compatible, and so on. If any of these legality checks fails, or if an error condition like overflow is encountered during the computations themselves, then it types an appropriate error-message.

Putting the legality checks into the library routines themselves was an almost inevitable decision. The author of a new library routine should not have to change programs that other people have written, and he is the person who knows what his routine can accept.

Building, Running, and Interrupting Processes

A process, like a routine, is an executable file. Actually, only a small piece of it is executable, a dozen instructions at the beginning, to which control is transferred when the process is called. These few instructions set up the process runner on the same map and transfer control to it. The rest of the process consists of a list structure that the runner interprets.

The process builder can be used to construct a new process, or to modify an existing one. It takes each statement from the APEX buffer area and incorporates the statement into a simple list structure, assigning it a line number unless the user has specified a number himself. These line numbers provide the necessary identification for replacing, inserting, and deleting lines, or for printing parts of a process. The builder checks each line to see whether it is an ordinary statement, a meta-command (e.g. to delete a line), a jump statement, parameter line, or an exit line. These last three types are checked for format, but other lines are inserted into the list structure without any format checking. Dummy pa-

rameters and names beginning with periods are noted for special treatment at run time.

In handling an ordinary statement, the process runner behaves very much like the basic translator. It must construct temporary names for names* that begin with periods, and it must substitute the appropriate terms for dummy parameters, but aside from that, it just builds the same calling sequence that the basic translator would and issues a go-up request. Like the basic translator, it does not know whether it is calling a library routine or another process.

And like the basic translator, the runner makes only those legality checks that cannot be left to the routines that it calls. It makes the same checks that the basic translator does, and it objects when it tries to execute a line that contains a dummy parameter for which no term was provided in the calling sequence by which the process itself was called. It also checks jump lines to determine that the jump is to a line that exists, and to determine that the relation in a conditional jump is a relation between scalars. And finally, when a routine that the runner has called peels back, the runner checks to see whether there is a flag indicating that the routine encountered an error condition. In this last case the routine will have printed an error message, so the runner does not. But in all cases the runner types the line number and the name of the process it is running. It then suspends the process and goes up to the basic translator on the next map.

There are two other ways in which a process may be suspended. The process itself may include a statement that explicitly says go up to the basic translator, and the user may press the interrupt button, which also has the effect of a go-up to the basic translator. In any case, once the process has been suspended, the user may use the basic translator to call for processes and routines. He may also give a special command that resumes the interrupted process, or a command that destroys the whole stack of maps and returns control to the basic translator on the lowest level.

New Primitive Routines

No matter how much effort is put into the public library, the user will sometimes have to write a new primitive routine, or get a programmer to do it for him. A compiler for a small procedure-oriented language is therefore being written with the help of a compiler-compiler⁷ that is under development at the

Laboratory. The object programs are named in the directory; they can be called in the normal manner by the basic translator or the process runner, can accept as inputs the files and numbers named in the directory, and declare their results in the directory—all as a library routine would.

The language is rather simple: it has no provision for text (i.e., literals), for in-out operations, for sub-routines, or for calls to other routines. It has an *arithmetic* statement, which is very like FORTRAN's, except that subscripts are actually typed as subscripts; a *begin* statement, which simply marks the beginning of a loop; an *end* statement, which marks the end of a loop and shows the ranges over which the variables in the loop are to run; a *jump* statement, which transfers control to another statement in the same routine; a *conditional* statement, which contains, first, a relation between two numbers, and second, an ordinary arithmetic or jump statement that is executed if the relation is true; and a *calling* statement that shows which names are to be treated as parameters and indicates the order in which they will appear in the calling sequence when the routine is used. There is also a *dimension* statement that gives the dimensions of an output array (the programmer does not declare the dimensions of input arrays). The dimension statement is executable: it generates code that requests APEX to create a file of a certain size.

An unusual feature of the language is a type of expression called a "greatest allowable subscript." For example, BETA~2 means "the second dimension (i.e., the number of columns) of the array named BETA." Expressions of this type may be used freely in dimension statements, arithmetic statements, and so on.

The seven statements listed above seem adequate for the user who wants a routine for his own use, but they are not really enough for the author of a new routine for the public library. He will need statements that can be used to check for error conditions—such as an undefined input term—and to specify the messages that he wants printed when various errors are detected.

Comments on the Distribution of Control

Perhaps the most common policy in designing a system for on-line use is to put a compiler, or some other language processor, in control of the calcula-

tions that the machine performs for the user. As has been seen, we adopted a different policy, which may be called a policy of distributed control.

The basic translator is a fairly simple routine, and can scarcely be said to control the system. In fact, no one routine is in charge. Consider for example that the basic translator may pass control to a process, which sets up the process runner, which may pass control to another process; the new process again sets up the runner, which may now pass control to a library routine, which will peel back to the runner; and so on. Throughout the whole chain of events, no routine that puts control into another map, either by going up or by peeling back, has any information about the nature of the routine that will take control. Thus, control is handed about among modules that are remarkably independent. They are held together only by the stack of maps, which records the return path through the structure, and by the directory, in which all results are entered so that other modules can find them.

Even the control of the time-sharing system itself is not vested in a translator in the usual way. APEX has no command language and no command language translator. The functions that are usually treated as commands—e.g., creating a synonym, dropping a file, resuming a suspended process—are all performed by small library routines that issue the necessary calls to APEX. The use of these routines is no different from the use of a matrix inversion routine, either from the user's viewpoint, or from the viewpoint of whatever routine calls them.

This way of handling commands is so neat and straightforward that it has much to recommend it, even in a system where control is not distributed in the more thorough sense described above.

We should mention that some of our colleagues feel there is one point at which we have not carried the distribution of control far enough. A routine that detects an error controls the delivery of the error message to the user: it just hands the message to APEX and requests that the message be printed. A more flexible arrangement would be to have the routine pass an error indication to the routine that called it. Then the calling routine could decide whether to print the message or to take some other action.

Advantages of Distributed Control

The virtues of distributed control arise mainly from the severe modularity that it enforces. The

same advantages could no doubt be obtained by other means, with enough persistence and enough administrative effort, but with distributed control they fall out naturally.

In the first place, there are advantages to the user. The system not only is modular; it looks modular. From the user's viewpoint the library routines are tools with stable, fairly simple properties. When the system comes to rest—i.e., when control is returned to the basic translator—he can use those tools to inspect his processes and his results before he embarks on another computational excursion. These same tools are available when the system comes to rest on a higher map after a process is suspended. The users therefore have not felt any great need for extra debugging tools.

Second, there are advantages to the system programmer. As usual, thorough modularity means that the system is generalized by making additions, not alterations. Even a new data-type—list structures, for example—is strictly an additive matter. The programmer must write library routines that will create and manipulate files of the new type, but beyond that his only problem is administrative: he must agree with other programmers on an identifying code for files of the new type. He does not have to make any change whatever in the existing software. There is also a peculiar advantage to the author of a new problem-oriented language. Assuming that the necessary library routines already exist, he can write a translator that just makes up a series of calls to those routines.

And finally, we expect there will be some sociological advantages that might be summed up by Licklider's⁸ phrase "coherent programming." He suggested that the software for an on-line system should not be created at one stroke; instead, he envisioned "a cooperative mode of programming through which a community of creative people, most of them substantive users, but some of them professional system programmers, might over a period of a few years develop a software base" of the sort that substantive users need. We would add that the basic requirement for coherence is this: The results of any operation must be ready for use as inputs to any other operation, even when the operations are described in different languages. This of course assumes that the operations make sense—the system is allowed to balk at computing the eigenvalues of a list of cities—but with that restriction, we would say that a soft-

ware base is coherent only to the extent that it meets this requirement.

An operation-oriented system, with the modularity enforced by distributed control, seems a natural environment for coherence of just that kind. Because adding to a modular structure is relatively easy, a programmer will often find it economical to write his program as a library routine, or as a translator that just calls on library routines. In any event, he will generally be able to save himself work by making his program compatible with the rest of the system. If his program can use results produced by other parts of the system, and if it stores its results in such a form that other parts of the system can use them, then the rest of the system will provide the user with many auxiliary services that the programmer would otherwise have to provide himself. That sort of incentive may be what is needed to make coherent programming more common than it has been. A chance to avoid extra work is often more effective than social pressure.

APPENDIX

The following is a list of the 64 operations available in the Reckoner public library in the summer of 1966. (There are not, however, 64 distinct routines; for example, sine and cosine are entrances to the same routine.)

Basic Arithmetic on Arrays (12 operations)

<i>On two arrays</i>	<i>On one array</i>
Add	Reciprocal
Subtract	Square root
Multiply	Log (base e or 10)
Divide	Antilog (base e or 10)
	Sine
	Cosine

Other Arithmetic on Arrays (8 operations)

- Add a scalar to all elements
- Multiply all elements by a scalar
- Find sums of rows (columns)
- Multiply each row (column) by corresponding element in a $1 \times n$ or $n \times 1$ array
- Give largest and smallest elements
- Find dimensions (number of rows, of columns, etc.)

Data Shuffling in Two-dimensional Arrays (14 operations)

- Copy an array, transposing it
- Copy an array, deleting certain rows (columns)
- Copy an array, saving only certain rows (columns)
- Copy an array, changing one element
- Copy an array, replacing one row (column) by a given $n \times 1$ or $1 \times n$ array
- Copy one element
- Copy two or more arrays, and join the copies to make a new large array
- Copy a square array, replacing the diagonal by a given $n \times 1$ or $1 \times n$ array
- Copy the diagonal and make it a $1 \times n$ or $n \times 1$ array

Matrix Operations (6 operations)

- Matrix multiplication
- Matrix inversion
- Rank
- Determinant
- Eigenvalues and eigenvectors of a symmetric matrix
- Solution of the matrix equation $AX = B$

Signal Analysis (2 operations)

- Fourier transform
- Inverse Fourier transform

Input (6 operations)

- Enter an array (or single number) from keyboard
- Create a $1 \times n$ array of the integers 1, ..., n
- Create an $n \times m$ array of zeros
- Define a new process
- Change a process

Output (9 operations)

- Type a scalar, an array, a row, a column, an element, text, or a process
- Type dimensions of an array
- Type largest and smallest elements
- Type a synonym of a given name
- Xerox listing of all names now defined
- Plot on CRT (with or without automatic choice of scales)
- Turn plot off

Miscellaneous (7 operations)

Define a synonym for a name
 Undefine a name
 Undefine all names of an entity
 Go up to the basic translator
 Resume operation at the next lower map
 Resume operation at the lowest map
 Prepare for sign-off

ACKNOWLEDGMENTS

C. K. McElwain and L. G. Roberts, together with some of the present authors, were responsible for designing APEX to support operation-oriented systems. M. L. Goodman, E. A. Martin, and M. L. Wendell were major contributors to the Reckoner public library. M. A. Morfield, R. T. Mitchell, and, especially, J. E. K. Smith contributed to the design of the facility as seen by the user.

REFERENCES

1. M. Greenberger, et al, "On-Line Computation and Simulation: The OPS-3 System," MIT Press, Cambridge, 1965.
2. R. Kaplow, S. L. Strong and J. W. Brackett, "MAP: A System for On-Line Mathematical Analysis," MIT Project MAC Technical Report No. 24 (Jan. 1966).
3. L. J. Wood, et al, "AMTRAN—Pushbutton Route to Instant Mathematics?," *Datamation* (in press).
4. G. J. Culler, "The Computer-Aided Lecturer," *Proc. Spring Joint Computer Conference*, Spartan Books, Washington, D.C., 1966.
5. —, and B. D. Fried, "The TRW Two-Station, On-Line Scientific Computer: General Description," in M. Sass and W. D. Wilkinson (eds.), *Computer Augmentation of Human Reasoning*, Spartan Books, Washington, D.C., 1965.
6. J. W. Forgie, "A Time- and Memory-Sharing Executive Program for Quick-Response, On-Line Applications," *Proc. Fall Joint Computer Conference*, Spartan Books, Washington, D.C., 1965.
7. J. A. Feldman, "A Formal Semantics for Computer Languages and its Application in a Compiler-Compiler," *Communications of the ACM*, vol. 9 (Jan. 1966).
8. J. C. R. Licklider, "Languages for Specialization and Application of Prepared Procedures," in Spiegel and Walker (eds.), *Second Congress on the Information System Sciences*, Spartan Books, Washington, D.C., 1965.

TELSIM, A USER-ORIENTED LANGUAGE FOR SIMULATING CONTINUOUS SYSTEMS AT A REMOTE TERMINAL

Kenneth J. Busch

*Bell Telephone Laboratories, Incorporated
Whippany, New Jersey*

INTRODUCTION

Telsim is a TEletypewriter SIMulation language designed for nonprogrammers and written for a moderate-size time-sharing system. It simulates continuous systems that can be described by a block diagram. Its input language is a natural engineering description of the boxes that comprise the diagram. The time-sharing system is offered commercially by the General Electric Company.¹ This system uses a GE-235 computer shared by a number of remote model 33 or 35 teletypewriters. The programming language is a special version of FORTRAN II.

Telsim is more than a problem-oriented language; it is also user-oriented. In the past many problem-oriented simulation languages have been written for a batch-processing environment.² Many of these have been only a limited help to nonprogrammers in their use of digital computers. Telsim's advantage lies in freeing the user from memorization of an instruction manual and from the requirements of specialized formats. The machine remembers the rules and helps the man. The man directs the computation; the machine does it. In this way a complex problem can be formulated and solved with Telsim in a relatively short interactive session.

TELSIM CHARACTERISTICS

During the problem definition stage, the boxes are specified one at a time in the order chosen by the user. He types the box number, its contents, and the inputs from other boxes. Each box is arbitrarily assigned a label from 1 to 99. A box may contain a signed or unsigned number, a symbolic constant, a function name, an algebraic operator, or an integration operator. All types are shown in box form in Fig. 1. When the block diagram is completely specified, Telsim produces (and types out upon request) a set of first-order differential equations that represent the dynamic behavior of the box graph. An auxiliary set of output equations is also compiled if required. These equations are valid Fortran expressions.

After seeing the equations, the user may recycle and edit the block diagram, deleting or adding boxes. In fact if he elects to save his input on a private file, he may resume this particular problem any time in the future. Once he is satisfied that his problem has been adequately described, the user directs Telsim to generate a Fortran simulation program that contains the equations for his system. He then requests the computer to compile the program and to run it with precompiled subroutines. The fixed portions are stored in the time-sharing system

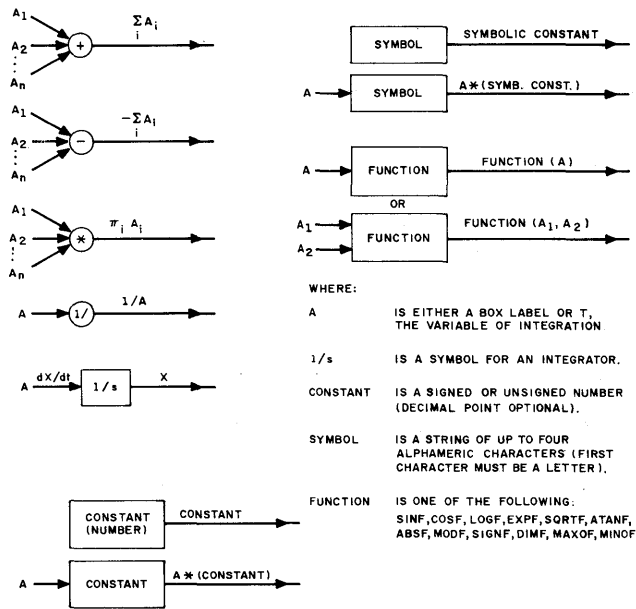


Figure 1. Box types.

and available on call. The user may also permanently store any simulation program produced by Tel-sim.

During the problem simulation stage, the user directs the computation. He may make any number of simulation runs. The simulation consists of numerically integrating the set of differential equations over a given interval and printing output at a specified period. The dependent variables in this set will be called state variables. The independent variable is designated as "T." Before and after each simulation, the user may list the values of state, change the state initial conditions, and indicate which state variables should be included in the printout. He may also list and change the values of any symbolic constants. The state variables and constants are listed and changed by index numbers. A request may be made to list the user-assigned names corresponding to these index numbers.

The user can also change the integration control parameters prior to a simulation run. These parameters are the initial and final value of T, the integration step size (ΔT), and the print period in units of T.

In addition, the simulation can be made to recycle automatically until a miss tolerance is met or a specified number of iterations has been completed. Both of these values are under the user's control. On each iteration a state initial condition will be incremented in a direction to minimize the miss.

This increment(s) is supplied by the user before requesting a run in the optimization mode. If on a given rerun of the simulation the miss increases, the initial condition will be incremented in the opposite direction and the run repeated. Should both directions fail to decrease the miss, the initial condition will be reset to its original value. Hence any iteration may recycle the simulation one or more times.

A successful iteration causes an adjustment of the initial condition of a state variable. An unsuccessful one leaves the initial condition for a given variable unchanged. This process is continued using successive state variables with nonzero increments until the recycling is terminated. If all variables with nonzero state increments have been perturbed and a termination criterion has not been reached, the entire process is repeated. The first state variable adjusted is again changed in the optimal direction (if any) found on the previous attempt. The user can select this optimization mode only if he designated a particular box output as the miss variable during the problem definition phase. Before an optimization run is started, the user indicates how often throughout the run he wishes to be given a choice of continuing or terminating the recycling.

AN ILLUSTRATIVE EXAMPLE

As an elementary example, let us suppose a bombing plane, flying at a constant altitude of 1024 feet with a velocity of 240 ft/sec, is overtaking a surface ship traveling at 80 ft/sec in the same direction as the plane. At what distance astern of the ship should a bomb be released in order to hit the ship if the air resistance is neglected? ³

Suppose in addition that just as the bomb is released the captain of the ship spots the plane through his telescope. If the plane appears to be very low on the horizon, he is unconcerned and allows his ship to proceed at its current speed. However, if the plane is almost overhead he orders the engine room to accelerate at 1 ft/sec.² Let us assume then, that in effect the captain of the ship continuously orders an acceleration equal to the sine of the angle that his telescope makes with the horizon. Under these conditions at what point should the plane drop the bomb?

This problem can be described in block diagram form as shown in Fig. 2. The blocks have been arbitrarily numbered. The user now calls the computer with the "hello" sequence shown in Fig. 3 and re-

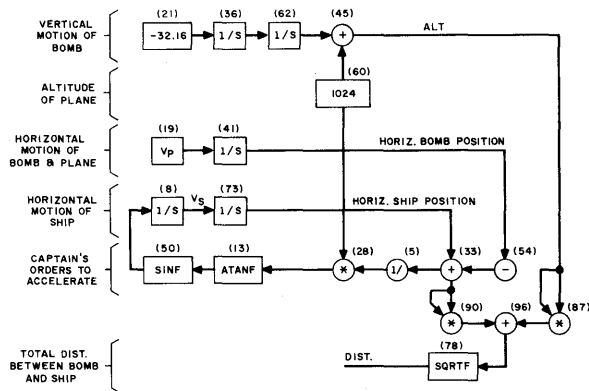


Figure 2. Typical block diagram.

requests an instruction manual on how to use the program.

Figure 4 shows the procedure for describing the block diagram to the computer. The sequence "BOX:=" is supplied by the machine; the user types the rest of the line. In this sample the boxes were described randomly to emphasize this feature. After all the boxes are specified, Telsim asks the user to identify those boxes (a maximum of five including T) whose outputs should be printed during the simulation run. He gives the box number and a title for each output. Finally, if the user wishes to use the optimization mode in the simulation program, he must indicate which output variable is to be optimized. This variable is called the terminal miss. The above procedure is shown in Fig. 5.

Telsim then asks the user if he wishes to see the equations for his problem. In problems that are initially formulated in terms of a block diagram an examination of the equations is an important part of the system analysis. Indeed for those problems described originally as equations (and converted to a block diagram) a comparison of the equations derived by Telsim with the original ones is a valuable check. The user may wish to change the block diagram after inspection of the equations. Telsim provides this opportunity. Once the user is satisfied with his input, he directs Telsim to punch out the program. The equations and a partial listing of the punched program for this example are shown in Figs. 6 and 7, respectively.

This program is then loaded with fixed portions of Telsim and the simulation started. In Fig. 8 the user requested a manual describing how to use the simulation. Figure 9 shows the method of initializing the integrators and specifying constants. In this

problem the initial conditions on state variables (ST.IC) have the following meanings:

Index No.	Symbol	Definition	IC
1	X8	Horizontal Velocity of Ship	80 ft/sec
2	X36	Vertical Velocity of Bomb	0 ft/sec
3	BOMB	Horizontal Pos. of Bomb	0 ft
4	X62	Change in Alt. of Bomb	0 ft
5	SHIP	Horizontal Pos. of Ship	(Unknown)

"VP" is the only symbol in this problem. It represents the horizontal velocity of the plane, 240 ft/sec. The user's strategy in finding the solution to this problem is to make the following two runs:

1. A standard simulation run in which the initial position of the ship is directly under the plane (0 ft).
2. An optimization run using the results of the first run.

The above strategy is arbitrary but it was chosen because it demonstrates an efficient use of interaction in order to converge quickly on a solution.

The first run, shown in Fig. 10, will yield the time it takes for the bomb to reach zero altitude. A good approximation to this time is found by hand interpolation between the last positive altitude and first negative altitude points. The second run is then made using this time as the simulation run time and the final separation distance near impact as the initial guess for the initial condition of state variable 5. State increment 5 is now set to 9 feet and an optimization run is made as shown in Fig. 11. The optimization proceeds until the final separation distance at impact is less than 10 feet. Listing the state initial conditions yields the answer that the plane should drop the bomb 1254 feet astern of the ship to score a hit.

TELSIM IMPLEMENTATION

A syntactic form is generated for each box as it is specified by the user. Once the block diagram is completed, a one-pass algorithm is used to compile from this information equations in infix notation with minimal parentheses. Some simplification in the form of the equations is achieved by using tables to control replacement of certain operator combinations. The algorithm permits inclusion of all Fortran functions without defining each function name as an operator. This is accomplished by introducing three special operators for functional notation. In fact this

SHELLO
 3 USER NUMBER--WHP051KJB
 SYSTEM--\$B FORTRAN
 RUN TYPE-- \$LOAD INPUT, GRAPH, OF, BOXES

LOAD LIMITS 10463 16057

DO YOU WANT AN INSTRUCTION MANUAL
 ANSWER YES OR NO:=YES

TELSIM ACCEPTS A DESCRIPTION OF A BLOCK DIAGRAM.
 EACH BOX IS INPUTTED AS FOLLOWS:

BOX:= NO TYPE INPUTS

WHERE BOX:= IS TYPED BY TELSIM
 NO IS A DISTINCT LABEL FOR BOX BEING INPUTTED
 TYPE IS DESCRIBED BELOW
 INPUTS ARE LABELS FOR THE LEADS INTO GIVEN BOX OR THE SYMBOL FOR
 TIME, T. INPUTS ARE SEPARATED BY ONE OR MORE SPACES.

A LABEL IS AN INTEGER FROM 1-99.

BOX TYPES ARE SPECIFIED AS FOLLOWS:

+ ALL INPUTS ARE SUMMED
 - ALL INPUTS ARE NEGATED, THEN SUMMED
 * ALL INPUTS ARE MULTIPLIED TOGETHER
 1/ INPUT IS DIVIDED INTO 1
 1/S INPUT IS INTEGRATED
 CONSTANT ANY SIGNED OR UNSIGNED NUMBER WITH OR WITHOUT A DECIMAL
 (OR FORTRAN FLOATING NOTATION)
 FUNCTION ANY ONE OF THE FOLLOWING FUNCTIONS: SIN F COS F LOG F
 EXP F SQRT F ATAN F ABS F INT F MOD F SIGN F DIM F
 MAX OF MIN OF (SEE GREEN CARD FORTRAN MANUAL, PP 14-15
 FOR DETAILS)
 SYMBOL ANY STRING OF FROM 1 TO 5 NON-BLANK ALPHABETIC OR
 NUMERIC CHARACTERS, AT LEAST ONE OF WHICH IS NON-NUMERIC.
 A SYMBOL MAY BE SIGNED OR UNSIGNED.

IF THE LEADS INTO A BOX ARE TOO NUMEROUS TO BE TYPED ON A SINGLE LINE
 THEY MAY BE CONTINUED ON THE NEXT LINE BY TYPING A \$ AS THE LAST
 NON-BLANK CHARACTER OF THE ORIGINAL LINE.

IN EDITING A BOX DIAGRAM, A BOX MAY BE DELETED AS FOLLOWS:

BOX:= NO DELETE

WHERE "NO" IS LABEL OF BOX TO BE DELETED

TO TERMINATE BLOCK DIAGRAM DESCRIPTION TYPE:

BOX:= END

Figure 3. Telsim instruction manual.

TO SAVE INPUT ON PRIVATE FILE OR RERUN USING BLOCK DIAGRAM DESCRIPTION ON PRIVATE FILE, ANSWER THE FOLLOWING REQUEST:

OPEN FILE NO. 4:=

WITH XXXXXX, NAMEYYYY

WHERE XXXXXX = YOUR USER NUMBER (FIRST 6 CHARACTERS)
 NAME = YOUR FIRST AND LAST INITIALS
 YYYY = YOUR PHONE EXT. (4 DIGITS)

IF SYSTEM TYPES "RETRY:=" REPEAT ABOVE INFO.
 IF SYSTEM TYPES "FILE NOT THERE, RETRY:=" YOUR OLD INPUT MAY BE LOST.

TELSIM ALLOWS THE USER TO SPECIFY UP TO 5 OUTPUTS.
 EACH OUTPUT IS SPECIFIED AS FOLLOWS:

OUT:= NO NAME

WHERE OUT:= IS TYPED BY TELSIM
 NO IS THE LABEL OF A BOX TO BE OUTPUTTED
 NAME IS A SYMBOL USED TO IDENTIFY THE OUTPUT (4 CHARACTERS MAX)

TO DESIGNATE TIME AS AN OUTPUT, TYPE

OUT:= T

TO DELETE THE LAST OUTPUT SPECIFIED, TYPE

OUT:= DELETE

TO TERMINATE OUTPUT SPECIFICATION, TYPE

OUT:= END

*** GOOD LUCK ***

Figure 3. (continued)

scheme can easily be extended to include any function as long as its skeletal form is available for embedding into the target code.

Syntactic Construction

After each input box specification is scanned for errors, the Polish suffix⁴ notation can readily be written. The Polish form for each type is given in Table I. All operators are either binary or unary. Unary operators are indicated with a subscript "u." Note that Hamblin's early-operator Reverse Polish form is used.⁵ In general, this should tend to reduce the total stack space required in the compiling algorithm. Function names, like symbols, are treated as operands. With this convention all functions are represented in terms of a binary and unary null operator (ϕ , ϕ_u respectively) and the comma operator. In tree notation, the function

$\langle \text{name} \rangle (\text{arg}_1, \text{arg}_2, \dots, \text{arg}_n)$

has the form indicated in Fig. 12. The null operators control the generation of parentheses as explained later.

The scanning and syntactic construction phase is shown schematically in Fig. 13. A pointer to the Polish information for each box defined by the user is placed in an index table. This box index table (LTB) consists of 99 locations, one for each possible box label. The information describing the Polish suffix form for a box is placed in the box Polish table (LTP). These two tables are used to feed a source stack as explained in the following section.

The final step in preparing the input for compilation is to set up two entries for each "output" box to be printed during the simulation. The output box number is entered into the output box table (LOB). In addition, a pointer to the user's output name is placed in the output name table (LON) at a location corresponding to the LOB entry.

The State and Auxiliary Equations

The desired equations are produced by a stack compilation technique. Before discussing this algorithm, first consider which equations must be generated. The output of each integrator is a state varia-

```
IS THIS A RERUN
ANSWER YES OR NO:=NO

DO YOU WANT INPUT SAVED ON A PRIVATE
ANSWER YES OR NO:=YES          FILE

OPEN FILE NO. 4:=WHP051,KB2003
```

```
BOX:= 21  -32.16
BOX:= 19  VP
BOX:=  8  1/S 50
BOX:= 50  SINP 13
BOX:= 36  1/S 21
BOX:= 41  1/S 19
BOX:= 73  1/S  8
BOX:= 13  ATANF 28
BOX:= 62  1/S  36
BOX:= 45  +  62 60
BOX:= 60  1024
BOX:= 28  *  60  5
BOX:=  5  1/  33
BOX:= 33  +  73 54
BOX:= 90  *  33 33
BOX:= 78  SQRTF 96
BOX:= 54  -  41
BOX:= 96  +  90 87
BOX:= 87  *  45 45
BOX:=END
```

```
TO CONTINUE TYPE  SLOAD OUTPUT,BOXES
*STOP           0 AT 11332  AFTER *STOP.
```

Figure 4. Input description of block diagram.

Table I—Polish Suffix for Each Box Type

Name	Designation	Polish Suffix Form
Summer	+	$B_1 B_2 + B_3 + \dots B_n +$
Negate Sum	-	$B_1 -_u B_2 - B_3 - \dots B_n -$
Multiplier	*	$B_1 B_2 * B_3 * \dots B_n *$
Reciprocal	1/	$B_1 1/_u$
Integrator	1/s	$DX_j B_1 =$
Unsigned Value	X.XX	$X.XX B_1 *$ (with input) $X.XX$ (no input)
Positive Value	+X.XX	Same as above
Negative Value	-X.XX	$X.XX -_u B_1 *$ (with input) $X.XX -_u$ (no input)
Symbolic Constant	<Symbol>	$\langle \text{Symbol} \rangle B_1 *$ (with input) $\langle \text{Symbol} \rangle$ (no input)
Function	<name>	$\langle \text{name} \rangle B_1 \phi_u \phi$ (with one input) $\langle \text{name} \rangle B_1 B_2, B_3, \dots B_n, \phi_u \phi$

Legend

B denotes a box and the subscript notation is as follows:

- 1, 2, ... n represents n inputs to a box
- i represents the only input to a box
- j represents the given box

A subscript "u" on an operator denotes a unary operator.

" ϕ " denotes a null operator.

ble. Let us designate these variables as X's. The input to an integrator is dX/dT , where T is the variable of integration. A set of state equations of the form

$$dX_i/dT = f_i(X_1, X_2, \dots, X_n) \quad i = 1, 2, \dots, n$$

SLOAD OUTPUT,BOXES

LOAD LIMITS 07625 16311

OUT:= T

OUT:= 73 SHIP

OUT:= 41 BOMB

OUT:= 45 ALT

OUT:= 78 DIST

OUT:=END

TO CONTINUE TYPE SLOAD COMPIL,EQNFOR,BOXES
***STOP 0 AT 06633 AFTER *STOP.**

SLOAD COMPIL,EQNFOR,BOXES

LOAD LIMITS 12407 15205

IS OPTIMIZATION DESIRED IN THE SIMULATION.
ANSWER YES OR NO:=YES

TYPE NAME OF TERMINAL MISS VARIABLE :=DIST

Figure 5. Specification of printout and optimization.

```

DO YOU WANT TO SEE YOUR EQUATIONS
ANSWER YES OR NO: YES

DX8=SINF(ATANF(1024/(SHIP-BOMB)))
DX36=-32.16
DOBOMB=VP
DX62=X36
DSHIP=X8
ALT=X62+1024
DIST=SQRTF((SHIP-BOMB)*(SHIP-BOMB)+(X62+1024)*(X62+1024))
TM=DIST

TO CONTINUE TYPE  LOAD PUNCH,PROG,FOR,BOXES  AFTER *STOP.
TO CHANGE YOUR PROBLEM, RERUN THE PREVIOUS SECTION BY TYPING,
LOAD INPUT,GRAPH,OF,BOXES
*STOP  @ AT 06412

```

Figure 6. Equations compiled by Telsim.

must be generated. Here i refers to successive integrators and is not the box number. There is a total of n integrators for a given problem. In addition to the state equations, auxiliary equations must be generated for each output variable that is not a state variable. These equations are of the form:

$$\langle \text{output name} \rangle = g_j(X_1, X_2, \dots, X_n)$$

Although not explicitly shown, the f_i 's and g_j 's may (and usually do) involve the values and symbols entered by the user. The equation for the input to an integrator (dX_i/dT) or an output variable can be constructed by threading backwards through the block diagram. This backward motion is always in opposition to the arrows of the diagram and is terminated on those branches that end in either a value, a symbol, or a state variable. For example, using Bn to represent the output of box n , the state equations for Fig. 2 are:

$$\begin{aligned}
 d(X36)/dT &= -32.16 \\
 d(X62)/dT &= X36 \\
 d(\text{BOMB})/dT &= V_p \\
 d(X8)/dT &= B50 \\
 &= \sin(B13) \\
 &= \sin(\text{atan}(B28)) \\
 &= \sin(\text{atan}(1024*B5)) \\
 &= \sin(\text{atan}(1024*(1./B33))) \\
 &= \sin(\text{atan}(1024*(1./(B73 + \\
 &\quad B54)))) \\
 &= \sin(\text{atan}(1024*(1./(\text{SHIP} + \\
 &\quad (-\text{BOMB})))))) \\
 d(\text{SHIP})/dT &= X8
 \end{aligned}$$

A similar relation can be written for the auxiliary equation needed to compute DIST. Note that if the user does not assign an output name to a state variable, the compiler must provide a distinct symbol to

represent it (Xk was chosen, where k is the box number).

The Compiler Algorithm

To generate these equations the box index and box Polish tables feed a source stack. This procedure is designed so that as the source stack is popped the Polish suffix for an equation is scanned from right to left. Scanning in this order permits identification of required infix parentheses in one pass. In compiling a state equation the source stack is primed with the Polish for the corresponding integrator box (i.e., $DX_j B_i =$).* The first token entered is the left-hand token of the Polish for the box (DX_j). The source is then popped on a last-in first-out basis (LIFO). Tokens representing nonintegrator boxes cause the source stack to be replenished by the corresponding box Polish form, left to right. A box token for an integrator box causes the state symbol (Xk or user output name if assigned) to be pushed down on the source stack.

Tokens not representing boxes or operators will be called "symbols" for the purpose of this discussion. Operators popped from the source stack are held temporarily in an operator stack. When symbol tokens are popped, they are entered directly into the target stack, preceded by any necessary right parentheses, and followed by an operator from the opera-

* For an auxiliary equation the Polish form

$$\langle \text{output name} \rangle B =$$

is used. B represents the box assigned to the output name.

```

      GO TO 100
00330 2  CONTINUE
00340  DX8=SINF(ATANF(1024/(OSHIP-OBOMB)))
00350  DX36=-32.16
00360  ODOBOMB=SVP
00370  DX62=X36
00380  ODSHIP=X8
00390  GO TO(101,102,103,104),IDE
00400 3  CONTINUE
00410  OALT=X62+1024
00420  ODIST=SQRTF((OSHIP-OBOMB)*(OSHIP-OBOMB)+(X62+1024)*(X62+1024)
00430  1))
00440  TM=ODIST
00450  PRINT: T, OSHIP, OBOMB, OALT, ODIST
YOUR PROGRAM TAPE HAS BEEN COMPLETED AND THE TAPE PUNCH SHOULD BE
TURNED OFF
DO YOU KNOW HOW TO PROCEED
ANSWER YES OR NO: NO
DO YOU KNOW HOW TO OPERATE THE PA
ANSWER YES OR NO: NO
WHAT MOD?

```

Figure 7. Partial listing of FORTRAN program compiled by Telsim.

LOAD SAMPLE, USING, TELSIM, SYSTEM

LOAD LIMITS 11552 16132

DO YOU WANT THE INSTRUCTION MANUAL.
ANSWER YES OR NO: =YES

TELSIM WILL SIMULATE YOUR BLOCK DIAGRAM SYSTEM. THE OUTPUT OF EACH INTEGRATOR (I/S) IS A STATE VARIABLE NAMED: XBB -- WHERE BB IS BOX NUMBER, OR ZZZ -- WHERE ZZZ IS YOUR OUTPUT NAME. IF YOU SPECIFIED SYMBOLIC CONSTANTS, EACH CONSTANT IS IDENTIFIED BY THE SYMBOL YOU GAVE.

LISTING NAMES AND VALUES: BEFORE AND AFTER A RUN, YOU CAN ASK FOR A LIST OF THE STATE VARIABLES AND SYMBOLS. IN LISTING NAMES AND VALUES AN INDEX NUMBER IS ALSO PRINTED. THIS INDEX NUMBER IS THE KEY THAT ASSOCIATES A VALUE WITH A STATE VARIABLE OR SYMBOL.

CHANGING VALUES: BEFORE A RUN, YOU CAN CHANGE ANY OF THE STATE VARIABLES OR SYMBOLS. AFTER A REQUEST FOR A CHANGE YOU MUST SUPPLY THE INDEX NO. IDENTIFYING THE ITEM TO BE CHANGED, A SPACE OR COMMA, AND THE NEW VALUE. AS MANY VALUES AS DESIRED MAY BE ENTERED ON SUCCEEDING LINES INCLUDING RETYPING A BAD ENTRY. WHEN FINISHED TYPE: 0 0

ASSOCIATED WITH EACH STATE VARIABLE ARE THE KEY WORDS:

ST.PRT -- PRINT FLAGS WHICH YOU CAN SET TO 1 IF YOU WANT THE INTEGRATOR VALUE PRINTED WITH THE SIMULATION OUTPUT.

SW.PRT -- SWITCH TO CONTROL PRINTING OF ABOVE INTEGRATORS. STATES ARE: OFF, NORMAL, INITIAL(ONLY), FINAL(ONLY).

ST.IC -- INITIALS CONDITION (I.E. THE VALUE) OF THE INTEGRATOR AT THE START OF THE SIMULATION.

ST.INC -- IF YOU SPECIFIED A TERMINAL MISS VARIABLE (TM) IN TELBOX, YOU CAN CHOOSE TO MAKE AN OPTIMIZATION (OPT) RUN IN WHICH THE SIMULATION WILL AUTOMATICALLY RECYCLE, AND THE ICS ARE SYSTEMATICALLY ADJUSTED UNTIL THE TM VALUE IS LESS THAN THE TOLERANCE YOU SPECIFY OR THE NUMBER OF ITERATIONS EQUALS THE MAXIMUM YOU SPECIFY.

ALL OF THE ABOVE CAN BE LISTED (INCLUDING NAMES), AND CHANGED AS DESCRIBED ABOVE. THE INDEX NO. OF THE CORRESPONDING STATE VARIABLE IS USED WHEN CHANGES ARE MADE.

Figure 8. Simulation instruction manual.

CONTROL OF INTEGRATION: YOU MUST SUPPLY (AND CAN CHANGE) THE INITIAL AND FINAL VALUES OF T, THE VARIABLE OF INTEGRATION. IN ADDITION, AN INTEGRATION STEP SIZE AND PRINT INTERVAL IN UNITS OF T MUST BE GIVEN. IF A PRINTOUT IS DESIRED ONLY AT THE BEGINNING AND END OF A SIMULATION MAKE THE PRINT INTERVAL EQUAL TO OR GREATER THAN THE TOTAL CHANGE IN T. WHEN MAKING AN OPTIMIZATION RUN YOU MUST SUPPLY (AND CAN CHANGE) THE TOLERANCE ON THE TERMINAL MISS, NO. OF ITERATIONS, AND THE NO. OF ITER. CYCLES BEFORE INQUIRY. ONE ITER. CYCLE CONSISTS OF AN ITERATION FOR EACH STATE VARIABLE WHOSE ST.INC IS NONZERO. AFTER THE NO. OF ITER. CYCLES SPECIFIED YOU WILL HAVE THE CHOICE OF CONTINUING OR TERMINATING THE OPTIMIZATION.

HOW TO TYPE NUMBERS:

INDEX NO. -- AN UNSIGNED INTEGER, NO DECIMAL POINT

VALUE -- A FLOATING POINT NUMBER OF THE FOLLOWING FORM:
 +1.2345678, -1234.5678, -123.45678E-2, +1.2345678E+24
 WHERE THE SIGNS, WHEN +, CAN BE OMITTED, THE DECIMAL MAY OCCUR ANY WHERE, THE NUMBER OF DIGITS MAY BE LESS, AND THE EXPONENT ESNN MAY BE OMITTED BUT WHEN PRESENT REPRESENTS THE POWER OF TEN WHICH MULTIPLIES THE VALUE.

ST.PRT FLAGS -- AN INTEGER, 0 OR 1

LEAVE A SPACE BETWEEN EACH NUMBER, HIT RETURN KEY WHEN FINISHED.

 ON COLD START (FIRST RUN) ALL ST.IC, SYMBOLS, ST.PRT, AND ST.INC HAVE BEEN CLEARED, I.E. SET TO ZERO. YOU MAY CHANGE THEM AS DESIRED.
 *****GOOD LUCK*****

Figure 8. (continued)

tor stack and left parentheses as required. The operator stack consists of two parallel stacks: the indirect operator and a left parenthesis counter. The flow of operator tokens and parenthesis information in and out of these stacks is the heart of the algorithm. The overall procedure is shown schematically in Fig. 14.

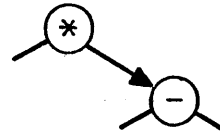
The token stream flowing across the interface in Fig. 14 is indeed the Polish suffix notation for the equation being generated. The first token across is the right-hand token of the suffix equation. This can be proven by the reader if he studies a simple block diagram and uses the standard Polish forms given in Table I. Insight into the details of the algorithm for transforming this stream into infix notation is gained by observing the dynamic building of equations using binary trees.

Let us consider a few simple algebraic terms before compiling a sample equation. The algebraic

terms $(A+B)*(C-D)$ are represented by the Polish suffix string:

$AB + CD - *$

When this is scanned from the right-hand side, the first two tokens can be represented diagrammatically, as:



The incomplete portion of the structure is indicated by buds on the nodes. The buds to the left and right on the “-” node will eventually point to the left and right operands, C and D respectively. The left bud on the “*” will blossom into a “+” node representing the left-hand term $(A+B)$. When the “-” node is added as the right-hand term to the “*” one can easily determine whether parentheses will be required. A parenthesis value for operators added to

TYPE VALUES FOR: INITIAL T, FINAL T, INTEGRATION STEP, PRINT INTERVAL
 :=0 10 .5 1

QUES: EXPLAIN LIST CHANGE RUN END :=LIST
 QUES: NAMES VALUES ST.PRT NONE :=NAMES
 QUES: SYMBOLS STATE ST.IC ST.INC NONE :=ST.IC

1 X8 2 X36 3 BOMB 4 X62 5 SHIP
 QUES: EXPLAIN LIST CHANGE RUN END :=CHANGE
 QUES: VALUES ST.PRT SW.PRT CONTROL NONE :=VALUES
 QUES: SYMBOLS ST.IC ST.INC NONE :=ST.IC
 :=1 80.
 :=0 0

QUES: EXPLAIN LIST CHANGE RUN END :=LIST
 QUES: NAMES VALUES ST.PRT NONE :=NAMES
 QUES: SYMBOLS STATE ST.IC ST.INC NONE :=SYMBOLS

1 VP
 QUES: EXPLAIN LIST CHANGE RUN END :=CHANGE
 QUES: VALUES ST.PRT SW.PRT CONTROL NONE :=VALUES
 QUES: SYMBOLS ST.IC ST.INC NONE :=SYMBOLS
 :=1 240.
 :=0 0

Figure 9. Method of identifying and changing variables and symbols.

QUES: EXPLAIN LIST CHANGE RUN END :=RUN
 QUES: STD OPT NONE :=STD

T	SHIP	BOMB	ALT	DIST
0.000000E-01	0.000000E-01	0.000000E-01	1.024000E+03	1.024000E+03
1.000000E+00	7.966768E+01	2.400000E+02	1.007920E+03	1.020593E+03
2.000000E+00	1.583492E+02	4.800000E+02	9.596800E+02	1.012149E+03
3.000000E+00	2.360781E+02	7.200000E+02	8.792800E+02	1.003650E+03
4.000000E+00	3.129036E+02	9.600000E+02	7.667200E+02	1.003291E+03
5.000000E+00	3.888839E+02	1.200000E+03	6.220000E+02	1.022151E+03
6.000000E+00	4.640802E+02	1.440000E+03	4.451200E+02	1.072638E+03
7.000000E+00	5.385523E+02	1.680000E+03	2.360800E+02	1.165606E+03
8.000000E+00	6.123562E+02	1.920000E+03	-5.120007E+00	1.307654E+03
9.000000E+00	6.855431E+02	2.160000E+03	-2.784800E+02	1.500525E+03
1.000000E+01	7.581592E+02	2.400000E+03	-5.840000E+02	1.742612E+03

QUES: EXPLAIN LIST CHANGE RUN END :=CHANGE
 QUES: VALUES ST.PRT SW.PRT CONTROL NONE :=CONTROL

TYPE VALUES FOR: INITIAL T, FINAL T, INTEGRATION STEP, PRINT INTERVAL
 :=0 7.98 .5 7.98

QUES: EXPLAIN LIST CHANGE RUN END :=CHANGE
 QUES: VALUES ST.PRT SW.PRT CONTROL NONE :=VALUES
 QUES: SYMBOLS ST.IC ST.INC NONE :=ST.IC
 :=5 1308
 :=0 0

QUES: EXPLAIN LIST CHANGE RUN END :=CHANGE
 QUES: VALUES ST.PRT SW.PRT CONTROL NONE :=VALUES
 QUES: SYMBOLS ST.IC ST.INC NONE :=ST.INC
 :=5 9.
 :=0 0

Figure 10. Standard simulation run.

QUES: EXPLAIN LIST CHANGE RUN END :=RUN
 QUES: STD OPT NONE :=OPT

TYPE VALUES FOR: TOLERANCE ON TERMINAL MISS, NO. OF ITERATIONS, NO. OF
 ITER. CYCLES BEFORE INQUIRY. :=10 10 4

T	SHIP	BOMB	ALT	DIST
0.000000E-01	1.308000E+03	0.000000E-01	1.024000E+03	1.661156E+03
7.980000E+00	1.970540E+03	1.915200E+03	1.916218E-02	5.533981E+01
T	SHIP	BOMB	ALT	DIST
0.000000E-01	1.317000E+03	0.000000E-01	1.024000E+03	1.668252E+03
7.980000E+00	1.979444E+03	1.915200E+03	1.916218E-02	6.424410E+01
T	SHIP	BOMB	ALT	DIST
0.000000E-01	1.299000E+03	0.000000E-01	1.024000E+03	1.654079E+03
7.980000E+00	1.961636E+03	1.915200E+03	1.916218E-02	4.643556E+01
T	SHIP	BOMB	ALT	DIST
0.000000E-01	1.290000E+03	0.000000E-01	1.024000E+03	1.647020E+03
7.980000E+00	1.952731E+03	1.915200E+03	1.916218E-02	3.753134E+01
T	SHIP	BOMB	ALT	DIST
0.000000E-01	1.281000E+03	0.000000E-01	1.024000E+03	1.639981E+03
7.980000E+00	1.943827E+03	1.915200E+03	1.916218E-02	2.862715E+01
T	SHIP	BOMB	ALT	DIST
0.000000E-01	1.272000E+03	0.000000E-01	1.024000E+03	1.632960E+03
7.980000E+00	1.934923E+03	1.915200E+03	1.916218E-02	1.972295E+01

DO YOU WANT TO CONTINUE
 ANSWER YES OR NO:=YES

T	SHIP	BOMB	ALT	DIST
0.000000E-01	1.263000E+03	0.000000E-01	1.024000E+03	1.625960E+03
7.980000E+00	1.926019E+03	1.915200E+03	1.916218E-02	1.081875E+01
T	SHIP	BOMB	ALT	DIST
0.000000E-01	1.254000E+03	0.000000E-01	1.024000E+03	1.618979E+03
7.980000E+00	1.917114E+03	1.915200E+03	1.916218E-02	1.914586E+00

QUES: EXPLAIN LIST CHANGE RUN END :=LIST
 QUES: NAMES VALUES ST.PRT NONE :=VALUES
 QUES: SYMBOLS STATE ST.IC ST.INC NONE :=ST.IC

1 8.00000000E+01 2 0.00000000E-01 3 0.00000000E-01 4 0.00000000E-01
 5 1.25400000E+03

QUES: EXPLAIN LIST CHANGE RUN END :=LIST
 QUES: NAMES VALUES ST.PRT NONE :=VALUES
 QUES: SYMBOLS STATE ST.IC ST.INC NONE :=STATE

1 8.6732114E+01 2 -2.5663680E+02 3 1.9152000E+03 4 -1.0239808E+03
 5 1.9171145E+03

QUES: EXPLAIN LIST CHANGE RUN END :=END

END OF RUN. GOOD-BYE.
 *STOP 0 AT 07556

ELAPSED TIME IN HUNDREDTHS OF HOURS 078

Figure 11. Optimization run.

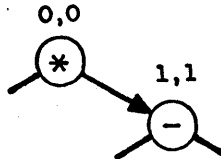
Table II—Right-Hand Table

Pointer	Indirect Op	Current Op										
		l/u	-u	β_u	β	,	/	*	+	-	=	
1	l/u	3	-1	NA	0	NA	NA	-1	-1	-1	NA	
2	-u	0	3	NA	0	NA	NA	0	-1	-1	NA	
3	β_u	1	-1	NA	0	0	NA	0	0	0	NA	
4	β	NA	NA	-1	NA	NA	NA	NA	NA	NA	NA	
5	,	0	0	NA	0	NA	NA	0	0	0	NA	
6	/	0	-1	NA	0	NA	NA	-1	-1	-1	NA	
7	*	0	-1	NA	0	NA	NA	0	-1	-1	NA	
8	+	0	0	NA	0	NA	NA	0	0	0	NA	
9	-	0	0	NA	0	NA	NA	0	-1	-1	NA	
10	=	0	0	NA	0	NA	NA	0	0	0	NA	

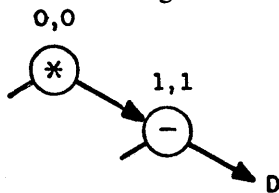
Legend

- NA: not applicable
- 0: no parentheses required
- 1: parentheses required
- >0: pointer to operator to replace indirect operator

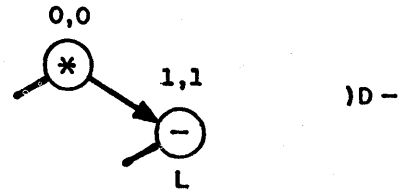
right-hand buds is found from Table II. For operators added to left buds, Table III is used. In these tables the added operator is the current operator; its parent in the tree ("*" in this case) is the indirect operator. The table look-up indicates parentheses are required. This is indicated as:



where the number on the left stands for a left parenthesis count; the right number, for a right parenthesis. The counts on the starting node are zero. The next token is then taken from the Polish string and added to the first available right bud:



As long as operators are being added the structure continues to grow. Each new operator is added to the lowest bud with a preference given to right buds. An operand, on the other hand, is a leaf of the tree. These leaves trigger the pruning of both operands and their associated branches. In pruning right branches the number of right parentheses indicated above the last operator is written into the target stack. The operand which triggered the pruning is written next, followed by the operator. The target stack resulting from the pruning is shown alongside



TREE TARGET STACK

the tree. The right side of this stack is the top of a LIFO stack. After pruning the right branch, the binary operator is marked with an L to indicate it has only a left bud remaining. This is necessary in order to prevent the operator from being written a second time when its left branch is pruned. The next token

Table III—Left-Hand Table

Pointer	Indirect OP (See Note)	Current op										
		l/u	-u	β_u	β	,	/	*	+	-	=	
1	l/u	0	0	NA	NA	NA	NA	NA	NA	NA	NA	
2	-u	0	-1	NA	NA	NA	NA	NA	NA	NA	NA	
3	β_u	0	-1	NA	NA	NA	NA	NA	NA	NA	NA	
4	β	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
5	,	0	0	NA	0	0	NA	0	0	0	NA	
6	/	0	0	NA	0	NA	NA	0	-1	-1	NA	
7	*	0	-1	NA	0	NA	NA	0	-1	-1	NA	
8	+	0	0	NA	0	NA	NA	0	0	0	NA	
9	-	0	-1	NA	0	NA	NA	0	0	0	NA	
10	=	0	0	NA	NA	NA	NA	NA	NA	NA	0	

Legend

- NA: not applicable
- 0: no parentheses required
- 1: parentheses required
- >0: Pointer to operator to replace indirect operator
- Note: When current operator is unary, the indirect operator is 1st "right" parent.

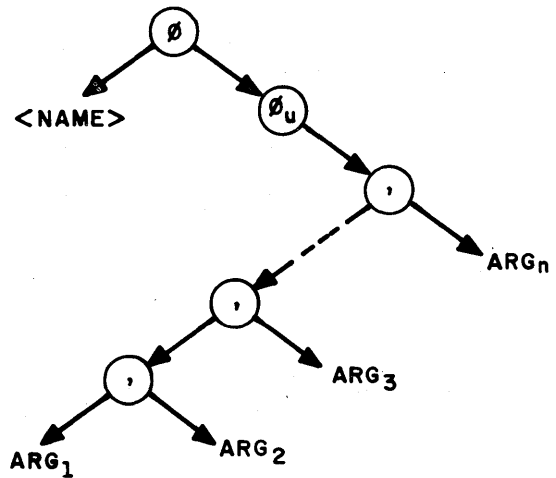


Figure 12. Function tree.

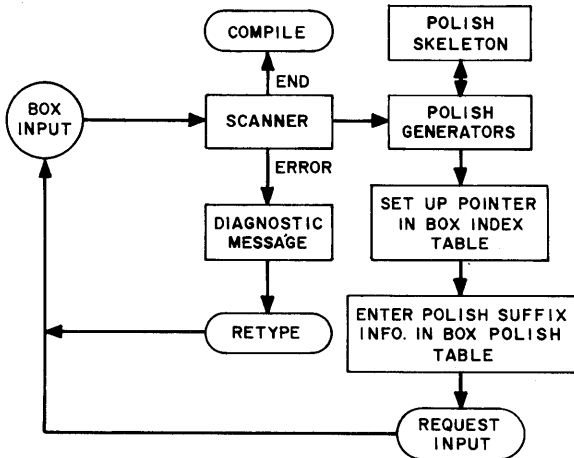
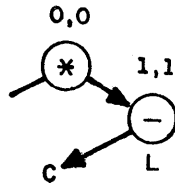


Figure 13. Syntactic construction.

C is added. In this case no right bud is available, so it is added to the lowest left bud of the structure.



In so doing, another branch is completed and pruning starts again. For a left branch the operand is written followed by the number of left parentheses indicated by the left count above the connecting node. The node is then discarded.



This also completes the right branch of the "*" node and causes it to be pruned and marked as described

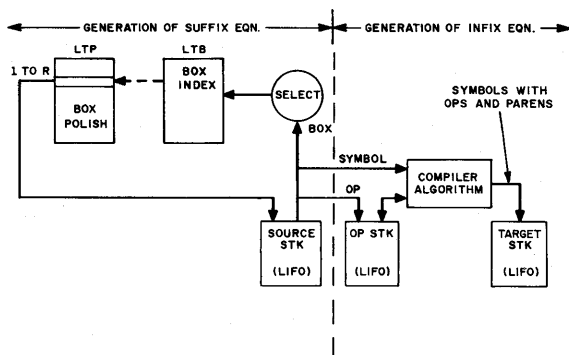
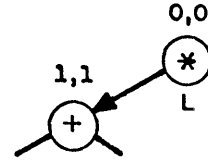


Figure 14. Compiling procedure.

above. Now the structure grows again with the addition of a "+" node to the remaining left bud.



In this case the parenthesis count is determined from Table III. The addition of the last tokens, in turn, completes the tree and causes pruning of all the nodes. The target stack will then contain:

$$)D - C(*)B + A($$

Unloading this on a LIFO basis gives the original terms in correct infix notation.

This simple example did not involve two very significant aspects in compiling infix equations. First, parentheses will accumulate as operators are nested within expressions. To accommodate this, the right-hand count is cumulative while the structure grows along successive right branches. This count is reset to zero when the growth changes to a left branch. Thereupon, the left count is accumulated as the structure continues to grow downward to the left.

Second, if the expression contains unary operators, slightly different rules are needed to accommodate them. If a unary term requires parentheses, for example in the expression $A*(-B)$, the left parenthesis is always adjacent to the operator. No amount of nesting of terms within an expression will ever break this bond. For a binary operator it would be to the left of the left-hand symbol. The corresponding right parenthesis for a unary operator, however, may be considerably removed from the operator. This occurs if the unary node is a parent of other nodes as in the expression $A*(-C*D/E)$.

This suggests that right parenthesis information for unary operators might be handled in the same fashion as that for binary. But the left parenthesis count should be treated differently. If the unary operator requires parentheses, the right counter is incremented and the operator itself is marked with a P. This mark will indicate that when the operator is written into the target stack, it should be followed immediately by a left parenthesis. With this convention unary nodes need carry only one count, the right parenthesis.

The expressions given above that involve unary operators should suggest that such operators may lead to situations where operator replacement is desirable. For example $A + (-B)$ is better written as

A—B. In this case the unary term is a right-hand term of the “+” operator. Note that Table II indeed indicates this replacement. A similar situation may prevail when the unary term is a left-hand term of a binary operator as in $A + (-B * C)$. Here $(-B)$ is the left term of “*”. But to ascertain that replacement is possible, one must consider not the immediate parent (the “*” in this case) but rather the closest “right” parent (the “+”). The left-hand table, Table III, shows this special consideration for these unary operators. Now consider the following equation:

Infix: $X = \text{Sin}((T - \text{TI}) * 2\pi * F)$

Suffix: $X \text{ Sin } T \text{ TI } -_u + 2\pi * F * \phi_u \phi =$

The unary minus operator was used, since the block diagram corresponding to this equation would cause it to be present. The dynamic growth and pruning of the tree that represents this equation is shown stage by stage in Fig. 15. The source and target stacks are shown at each stage.

The reader may want to follow the flow indicated in the successive parts in Fig. 15. Note that the nodes “+ -_u” were placed by “-” in Figs. 15(c) and (d). The target stack in 15(g) is completed when “X” is added as the last item. Unloading the target stack on a LIFO basis and suppressing the null operators (squeezing out blanks) gives the desired infix equation, left to right.

The complete algorithm is flow-charted in Fig. 16. Before compilation of each equation, the indirect operator (OP) and left parenthesis counter (Lparen) stacks are primed with “L=” and “O”, respectively. The purpose of this priming is to provide an indirect operator for the first operator in the source stack (always =) that gives an initial count of 0, 0. Note that the marking of nodes is indicated by preceding the operator with the marking symbol. Let’s step through the algorithm using the preceding suffix equation. This is shown in Table IV. The first line represents the priming operation. Succeeding lines give the results as successive tokens are taken from the source and processed. Popping the operator stack in the algorithm also implies discarding the Lparen entry. The last entry in Table IV shows the desired infix equation in the target stack.

Data Structure

A good data structure is important if compilation is to be fast. The structure used in Telsim permits quick identification of the three types of tokens:

Table IV—Compilation Example

Source Token	Counters		Stack	Contents of Stacks After Each Token Is Analyzed
	k	n		
(Prime)			Op Lparen Target	L= O
=	0	0	Op Lparen Target	L= = O O
ϕ	0	0	Op Lparen Target	L= ϕ O O O
ϕ_u	1	-1	Op Lparen Target	L= $\phi \phi_u$ O O O
*	1	0	Op Lparen Target	L= $\phi \phi_u *$ O O O O
F	0		Op Lparen Target	L= $\phi \phi_u L^*$ O O O O) F *
*	0	0	Op Lparen Target	L= $\phi \phi_u L^* *$ O O O O) F *
2π	0		Op Lparen Target	L= $\phi \phi_u L^* L^*$ O O O O) F * 2\pi *
+	1	-1	Op Lparen Target	L= $\phi \phi_u L^* L^* +$ O O O O O O 1) F * 2\pi *
- _u	1	9	Op Lparen Target	L= $\phi \phi_u L^* L^* -$ O O O O O O 1) F * 2\pi *
TI	0		Op Lparen Target	L= $\phi \phi_u L^* L^* L^-$ O O O O O O 1) F * 2\pi *) TI -
T	0		Op Lparen Target	L= $L\phi$ O O O O) F * 2\pi *) TI - T (ϕ_u (ϕ
Sin	0		Op Lparen Target	L= L= O O O O) F * 2\pi *) TI - T (ϕ_u (ϕ sin
X	0		Op Lparen Target) F * 2\pi *) TI - T (ϕ_u (ϕ sin - X

boxes, symbols (or function names or values), and operators. The entries in the box Polish table are actually pointers to the primitive information stored in a text table. The text table is appended to the box index table starting at location 100.* Symbols and values are then distinguishable from boxes since the latter have pointers whose magnitude is less than 100. To separate this operand class from the operators, the box and symbol pointers are made negative. In processing the operators, the algorithm also needs to identify its type: either binary or unary. This is accomplished by constructing the operator entries to point to an operator table (LTO). The operator table entry then references the primitive in the text table. A positive pointer in the operator table indicates that the operator is binary; negative is unary. Furthermore, the operator table is ordered to

* In this version of Telsim a maximum of 99 boxes can be specified.

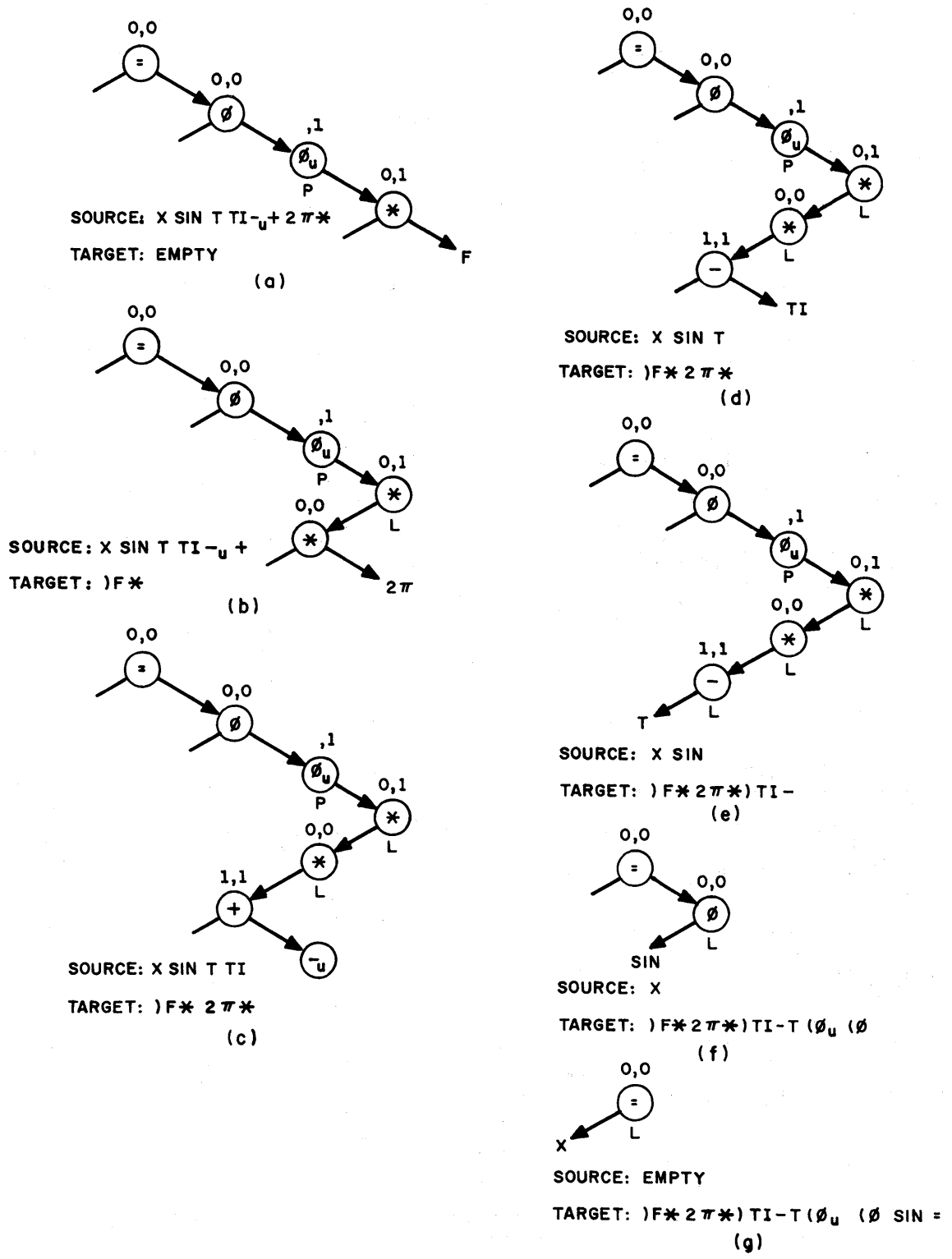


Figure 15. Dynamic tree structure.

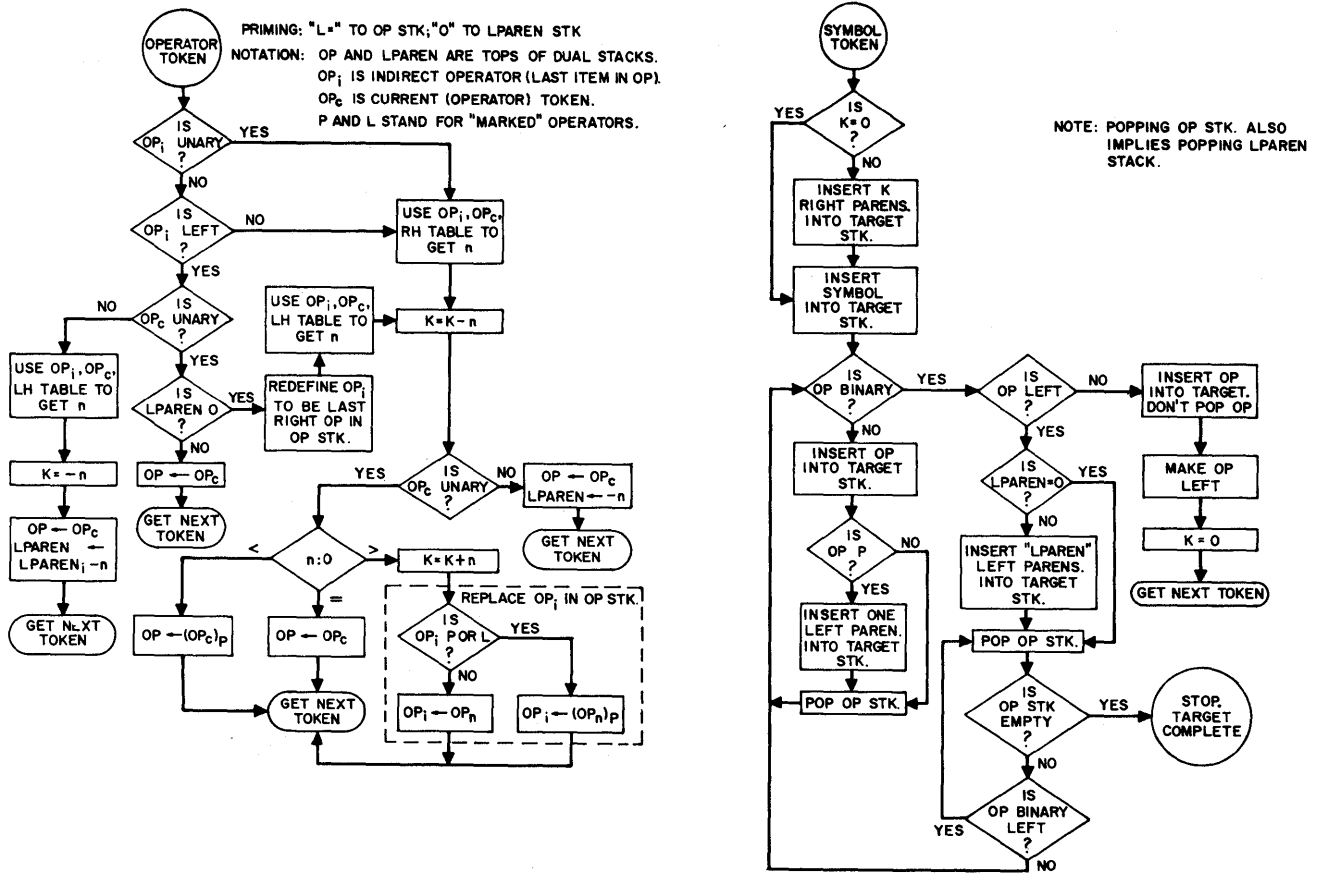


Figure 16. The compiler algorithm.

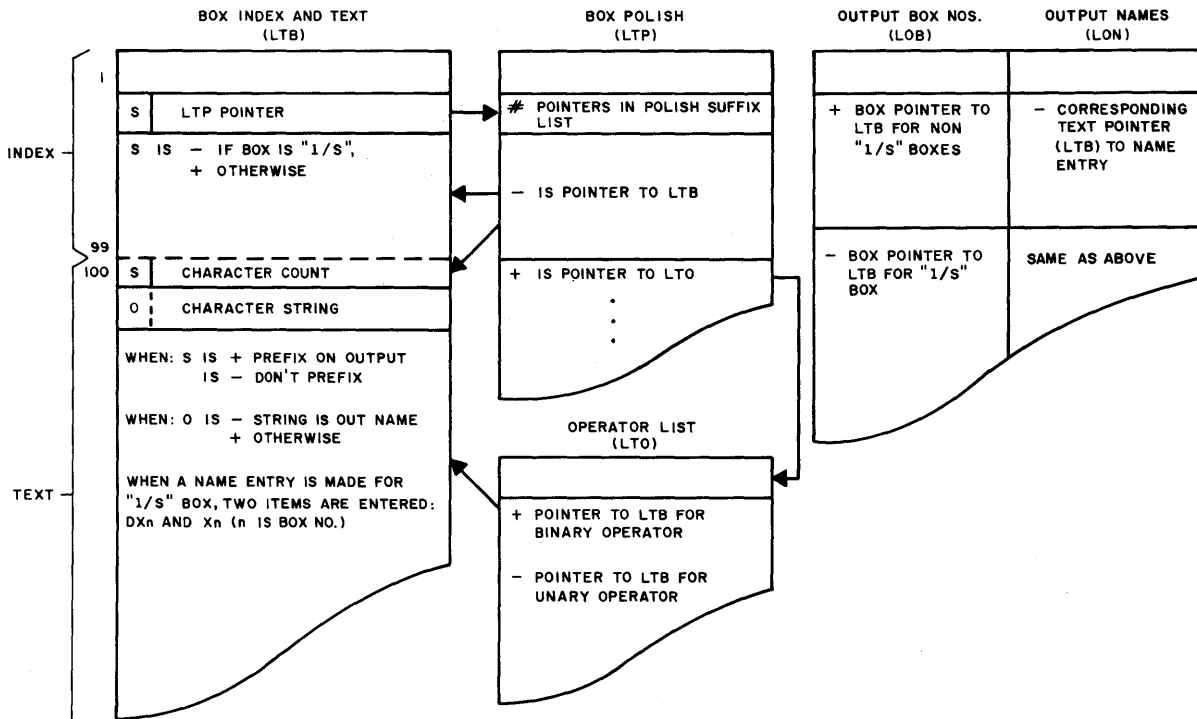


Figure 17. Telsim data structure.

correspond to the entries of the right- and left-hand tables used in the algorithm. The details of this structure are shown in Fig. 17.

Referring back to Fig. 14, we see that during the generation of the suffix equation the box pointers are numbers in the range of -1 to -99 . All symbol pointers are less than this, and operator pointers are positive. In the next phase, the algorithm proper, all symbol pointers are set positive when fed into the target stack. In pruning operators, the magnitude of the pointer in the operator table is placed in the target stack. This means that after an equation is compiled it will be a list of pointers to the primitives in the text table. Special conventions within the text table have been designed as seen in Fig. 17 to distinguish user symbols and output names from each other and from numbers and compiler-generated symbols. This permits symbol prefixing to be done on the fly in punching out the final program. No prefixing is done when they are printed for the user's inspection. These details will not be discussed here.

Before leaving this subject it should be mentioned that entries for integrators in the box index table and output box table are set negative. Negative pointers in the box index table indicate when a state variable is encountered in popping the source stack. Output integrator boxes also are marked, since no auxiliary equation will be needed for them.

Structure of the Simulation Program

The state and auxiliary equations are embedded by the compiler in a segment of a Fortran program. The complete simulation program consists of this variable segment together with a fixed segment and precompiled subroutines. The structure of the simulation program is relatively straightforward and is shown in Fig. 18.

The variable segment contains the necessary bookkeeping (common, equivalence, etc.) that changes from problem to problem. These instructions are arranged so that, together with flags that give the number of integrators and symbols, the common layout is known by all subroutines. Routines such as those that print and read the contents of various arrays must be able to communicate with this common data region. Following the "bookkeeping," a statement transfers control to the section that plays the "question game" with the user. The user is given a choice of actions for the program to carry

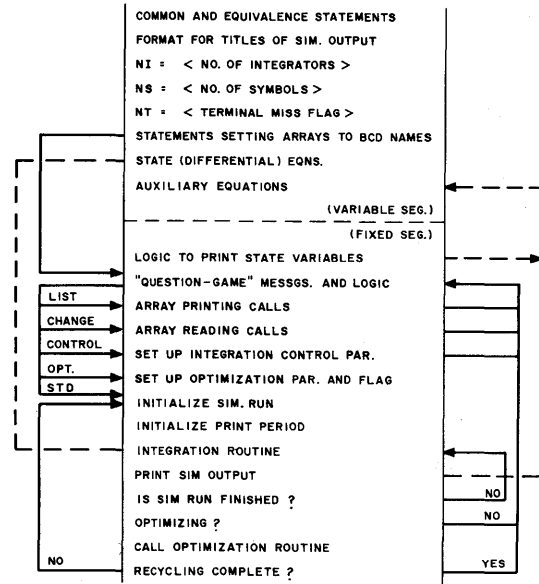


Figure 18. Simulation program structure.

out, such as list or change. Based upon his response, the next choice is funneled to a subset of such meaningful descriptors as names or values. This "funneling or steering" is a technique that has been employed in control of graphical routines. In this instance the user employs a light pen to point at the appropriate control word displayed on the cathode ray tube.⁶ Programming this interactive dialogue was facilitated by using a general purpose message routine.⁷

The variable segment is completed by embedding the state and auxiliary equations immediately after the statement transferring control to the "question game." The equations are executed, as indicated by the dotted lines in Fig. 18, by transferring out of (and then back to) the integration and print sections of the fixed segment. The integration method presently used is 4th order Runge-Kutta.⁸

The unusual structure of this program deserves comment. The GE-235 computer has a machine cycle of six microseconds and no floating-point hardware. This together with the time-slicing required to time-share the system tends to make large blocks of computation relatively slow. The simulation program was designed to minimize transfers to subroutines once the integration computations were under way. In effect, the integration and equations which compute the derivatives are the main program in Telsim. This corresponds to turning the usual structure of a simulation program "inside-out" in order to make it as fast-running as possible.

SUMMARY

This version of Telsim is an initial effort to provide user-oriented languages for control system analysis and synthesis. A user-oriented language must be responsive. There should be a man-machine dialogue during both problem definition and solution. Telsim attempts to do this.

Telsim is a compiler not an interpreter. It is a language programmed in Fortran that produces Fortran code designed for efficient simulation. Telsim uses a one-pass table-driven compiler algorithm to produce the state (differential) and output equations. This is a departure from the existing "sequencing" method used in current simulation languages.⁹⁻¹¹ "Sequencing" or "sorting" of boxes requires that an ordered list of Fortran expressions, one for each box, be generated such that the last expressions give the values for each of the derivatives. As a consequence of this new method there is no need to apportion storage space among the various types of boxes as done in the past. The user can specify any number of the different types as long as storage is not exhausted. He can then view the equations in meaningful form and thereby check his block diagram. These equations are valid Fortran expressions.* Telsim compiles efficient and fast-running Fortran code for either simple or complex diagrams. This results in a program comparable to one hand-coded by a good programmer.

By using the procedures presented in this paper, Telsim can be extended to allow the user to define and name functions by a block diagram composed of primitive boxes. In addition, Telsim has advanced language features such as symbolic labeling of constants, use of Fortran functions (including nesting), and relative freedom in input formats. The simulation program, itself, has an optimization mode that permits automatic adjustment of initial conditions in solving boundary value problems.

This effort has suggested a number of improvements. Three things will be done immediately. Little more can be, since this version of Telsim has taxed the available time-sharing system to its capacity. First, the optimization ability will be extended to include automatic adjustment of symbolic constants in an analogous fashion to the initial condition feature. Included with symbolic constants are the initial and final values of T , the variable of integration.

* The time-sharing system's Fortran compiler converts these into machine code.

Second, Telsim will be improved to allow the user to specify a termination function (akin to the terminal miss variable) to control the final time in a simulation run. And finally, the user will be provided a choice of the method of numerical integration to be used during the simulation run.

There are many more desirable features that one would like in a general simulation language. During problem definition, the following should be considered:

1. The ability to accept equations as the specification for the whole system or part of a block diagram.
2. The ability to simulate composite systems with both continuous and discrete portions. (For discrete system simulation the reader is referred to the BLODI language.)^{12, 13}
3. A larger subset of primitive boxes to specify nonlinear elements, transfer functions, random noise generator, and tables.
4. A simple method for the definition of new functional boxes thereby providing system growth.
5. Superblock specification by the user of those portions of a system that are repeated a number of times.
6. Graphical input specification.
7. Diagnostics during the compilation algorithm and efficient handling of sub-expressions found to be common.

During the simulation phase the user should have:

1. The ability to select from a set of optimization procedures.
2. The ability to specify events and criterion information.
3. Automatic determination of integration methods and step size.
4. More freedom in selecting printout (choosing any variable and nonuniform printing interval during a run).
5. Graphical output.

Needless to say implementation of these features will have to await the arrival of a larger time-sharing system. However, exploratory programming and development along these avenues will be started in a batch-processing environment.

ACKNOWLEDGMENT

The skillful programming done by Misses Mary Lou Flynn and Ruth L. Salmon, and Messrs. Robert E. Downes and Gardner C. Patton has made this version of Telsim possible. The helpful suggestions and comments by Dr. W. C. Ridgway, III during the preparation of this paper are appreciated.

REFERENCES

1. *Desk Side Computer System Reference Manual*, General Electric Company, Missile Space Division, Valley Forge, Pa. (Jan. 1966).
2. J. J. Clancy and M. S. Fineberg, "Digital Simulation Languages: A Critique and a Guide," AFIPS Volume 27, *Proceedings, 1965 Fall Joint Computer Conference*, Spartan Books, Washington, D.C., 1965, pp. 23-36.
3. F. W. Sears and M. W. Zemansky, *College Physics*, Addison-Wesley Press, Inc., Reading, Mass., 1948, p. 99, prob. 9.
4. P. Wegner, *Introduction to System Programming*, Academic Press, New York, 1964, pp. 101-121.
5. C. L. Hamblin, "Translation to and from Polish Notation," *The Computer Journal*, vol. 5, no. 3, pp. 210-213 (Oct. 1962).
6. W. H. Ninke, "Graphic I—A Remote Graphical Display Console System," AFIPS, Volume 27, *Proceedings, 1965 Fall Joint Computer Conference*, Spartan Books, Washington, D.C., 1965, pp. 839-846.
7. G. C. Patton, "The MESSAG Dialogue System," Bell Telephone Laboratories Memorandum (May 10, 1966).
8. F. B. Hildebrand, *Introduction to Numerical Analysis*, McGraw-Hill Book Co., New York, 1956, pp. 236-239.
9. Clancy and Fineberg, *op. cit.*
10. M. L. Stein, J. Rose, and D. B. Parker, "A Compiler with an Analog-Oriented Input Language," *Proceedings, Western Joint Computer Conference*, San Francisco, 1959.
11. R. M. Janoski, R. L. Shaefer, and J. J. Skiles, "COBLOC—A Program for All-Digital Simulation of a Hybrid Computer," *IEEE Transactions on Electronic Computer*, vol. EC-15, no. 1, pp. 74-82 (Feb. 1966).
12. J. L. Kelly, Jr., C. Lochbaum, and V. A. Vyssotsky, "A Block Diagram Compiler," *Bell System Technical Journal*, vol. 40, pp. 669-676 (May 1961).
13. B. J. Karafin, "The New Block Diagram Compiler for Simulation of Sampled-Data Systems," AFIPS, Volume 27, *Proceedings, 1965 Fall Joint Computer Conference*, Spartan Books, Washington, D.C., 1965, pp. 55-61.

MAN-MACHINE COMMUNICATION IN ON-LINE MATHEMATICAL ANALYSIS *

R. Kaplow, J. Brackett and S. Strong

*Massachusetts Institute of Technology
Cambridge, Massachusetts*

INTRODUCTION

During the past four or five years increasing attention has been paid to providing efficient, direct access to digital computing machines. Interactive or "on-line" systems have been implemented on various machines with a variety of programming techniques. Whether the system uses a small computer which can handle only one user-operator or is part of a time-shared multiple-access computing utility, the goal is to permit the user to have direct communication with an operating program. Among other advantages, such an on-line system permits a user to insert decisions during the course of problem solution, with his judgments benefiting from current results.

A large multiple-access system serving a wide variety of users, such as the Project MAC System,^{1,2} tends to become a repository for problem-solving techniques. Therefore, the lay user of such a system, in contrast to an experienced programmer, may more often be concerned with the problem of locating and learning to use suitable procedures than with writing a program himself. In such a circumstance, all of the problems associated with obtaining a pro-

gram from a library are present, including incomplete or nonexistent cataloging and documentation.

It appears that efficient general utilization of computers will depend on the evolution of subsystems which combine available procedures for specific applications in such a manner that little or no library searching or programming will be required. Such subsystems have already been developed in a number of specific fields, such as civil engineering,^{3,4} electrical circuit⁵ and nuclear reactor design,⁶ and simulation techniques.⁷ It also appears to be a feasible approach in the more general area of mathematical analysis since appropriate combinations of approximately 30 to 50 numerical procedures will suffice for large classes of problems. Though users may ultimately do relatively little programming with such a system, it is clear that rather sophisticated programming will be required to support the system.

The present paper, which deals with a new system for on-line mathematical analysis, illustrates a general approach which can eliminate programming for many users. Our interest in the program grew out of consideration of the manner in which digital computation techniques could be best utilized in teaching subjects in which side excursions for programming could not be tolerated and in which "black box" programs would serve little or no purpose. It was decided, at the outset, that one would want to preserve an analytical approach, that is, the breakdown

* The work reported herein was supported by Project MAC, an MIT research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01).

of the problem into its component parts, each part consisting of a particular mathematical procedure. The required system would be one in which the writing (or, more likely, the typing) of normal mathematical nomenclature would cause the results of the operations to be generated. Thus it would be required that the student, or other user, have a sufficient grasp of a particular problem so that he could describe it in standard mathematical form. He would then expect the system to request all of the necessary specific information, do all the indicated mathematics and, on request, to present the results in a useful form. With these intentions, we have been developing such a system, called MAP (for Mathematical Analysis Program),⁸ for use in conjunction with the MIT Computation Center and Project MAC Compatible Time-Sharing Systems (CTSS).

Techniques for utilizing direct computer access in mathematical analyses have been discussed by J. C. Shaw,⁹ G. Culler and R. Huff,¹⁰ and G. Culler and B. Fried.^{11,12} In each instance the system was implemented within a less powerful facility than is available at the MIT Computation Center and Project MAC. CTSS facilitates the use of English language communication and allows the storage of extensive programs and data. The aims outlined in the foregoing paragraphs are therefore more readily implemented.

The most striking aspect of an on-line system is the extent to which it may be programmed to carry on a meaningful dialogue. While the computer cannot generate ad-lib responses, a sophisticated programmer, working within the framework specified by a particular type of problem, can anticipate what kinds of questions will be asked and what kind of responses will be made by a user. He can therefore pre-store or provide a program to generate responses to be made and questions to be asked in reply to the user's input.¹³ Most likely, of course, he will place a definite limit on the vocabulary which the computer will understand, simply in order to make the programming and storage requirements less demanding. The computer's responses can nonetheless be quite varied; they can depend on previous dialogue, the sophistication of the user's questions and responses, the results of calculations and other factors as well.

A number of previous applications have taken good advantage of this "conversational" facility of

direct-access computers. Among those which are in operation are systems for computer-aided design in mechanical engineering,¹⁴ design of electrical networks,¹⁵ searching the technical literature,¹⁶ describing and analyzing the three-dimensional structure of large molecules,¹⁷ and computer-aided teaching.¹⁸ For those who have had no previous contact with user-machine dialogues a very readable general discussion by Licklider will be of interest.¹⁹

The form and language of the user-machine communication is the most obvious and probably the most important single aspect in the design of a conversational system. It is necessary to strike a balance between the terminology commonly understood and an efficient but cryptic code. At one extreme is the possibility of using English prose throughout, and at the other the possibility of a two or three-letter code just complex enough to eliminate ambiguities. The former choice requires a great deal of rather useless translation and elimination of redundancies by the computer and an excess of input and output. On the other hand, codes are a nuisance for the user to memorize or decipher. The major requirement, of course, is that the demands on the user should be minimized, in terms of both typing and deciphering effort. We have therefore chosen neither of the extremes, but a combination of modifications of both. The user "talks" in one or two-word phrases or in arithmetic equations, while the computer uses a passable form of English. Conversation between the user and the computer consists primarily of questions and answers, with statements (for informational purposes) and outright commands occasionally admixed. Questions, which in the present context are any requests for information, are often phrased in the normal form of commands, because of the simpler grammatical form of the latter.

ILLUSTRATION OF USER-MACHINE COMMUNICATION

Even if space permitted, there would be little purpose in describing all of the facilities available within MAP. Rather we shall attempt to illustrate the salient features of the user-machine conversation with a specific illustration. For this purpose we select the following problem:

Given an experimental function, $\text{spect}(\nu)$, determine, by fitting, the best set of values for the physical

parameters q_i , v_i and Γ_i , the pertinent relationships being:

$$\text{spect}(v) = \int_{-\infty}^{+\infty} I(v') \text{ABS}(v-v') dv'$$

where $I(v) = \Gamma_s / (\pi(4(v^2) + \Gamma_s^2))$

$$\text{ABS}(v) = 1 - \exp \left[-b \sum_{i=1}^3 q_i \Gamma_i^2 / (4(v-v_i)^2 + \Gamma_i^2) \right],$$

and $\Gamma_s, b =$ known constants of the experiment.

The physical basis of the problem happens to be Mössbauer γ -ray absorption spectroscopy, but that is of no importance in this discussion. It may be perceived, however, that since the unknowns determine the positions, magnitudes and widths of a series of Lorentzian peaks in an absorption spectrum which is subsequently convolved with the source spectrum, the usual programming approach (in a language such as FORTRAN) will be quite complex. Even a discussion of a suitable program would require an inordinate amount of time if students were to utilize the computer in a laboratory subject. With MAP, as we shall see, it is only necessary for the student to understand the mathematics of the analysis at the same level that the instructor would normally desire.

If this were a laboratory problem, the instructor would probably suggest that the student compare his normalized data—to be called $\text{normda}(vr)$ —to trial calculations of $\text{spect}(v)$, until agreement was achieved and the corresponding parameters determined.

We will assume that the user has available to him one of the more than 100 remote terminals on the MIT campus, with or without display hardware. The terminals are connected to the computer system via a private branch exchange; a user may dial either the Project MAC or the Computation Center system. Once logged into the system, the user reaches MAP by typing "resume map" to the CTSS supervisor; the computer responds "COMMAND PLEASE", indicating its readiness for a command.

In this instance the first step would be to convert the experimental counting rate data into normalized absorbed intensity values. In the following example, note that the upper-case type is computer-generated; only the lower-case lines and the numerical input were typed by the user. The user inputs have been underlined to make them distinct.

COMMAND PLEASE

(normda(vr) = 1-cr(vr)/cbase)

DECIMAL VALUE OF THE CONSTANT CBASE
PLEASE. 5.544

CR(VR) IS NOT DEFINED. IF IT HAS A DIFFERENT NAME, TYPE THE NAME. IF YOU WANT TO TYPE IN NUMERICAL VALUES NOW, TYPE THE WORD INPUT. OTHERWISE, GIVE A CARRIAGE RETURN AND DEFINE THE FUNCTION BEFORE USING THE NAME AGAIN.

input

PLEASE PRINT ON THE NEXT LINE MIN, MAX, AND DEL FOR THE VARIABLE VR.

-5. 5. .1

MIN = -50 MAX = 50

TYPE IN DATA IN ARBITRARY FORMAT, EACH DATA POINT SEPARATED BY A SPACE. THE INPUT DATA CAN BE EDITED BY USING THE CONVENTIONS GIVEN IN THE MANUAL. WHEN ALL DATA POINTS HAVE BEEN ENTERED, GIVE TWO CARRIAGE RETURNS, COMPLETE EDITING IF NECESSARY AND GIVE COMMAND 'FILE INPUT DATA'.

INPUT:

5.544	5,544	5.549	5.545	5.546
5.549	5.551	5.542	5.547	etc.

EDIT:

locate 5,544

change /, /, /

print

5.544	5.544	5.549	5.545	5.546
-------	-------	-------	-------	-------

file input data

COMMAND PLEASE

As illustrated here, MAP will always request values for variables specified in an operation, but not previously defined. When arrays of data (i.e., functions) are requested, the input is placed under the supervision of an editing program and is available

for editing in Hollerith (BCD) form until written over by the next input data. The editing illustrated above, which was required because of the accidental and initially unnoticed typing of a comma instead of a decimal point, is self-explanatory. The editing facility is essentially identical to that available directly in CTSS.² In addition to the standard type-written form of data input, as illustrated above, the data can be inserted off-line in punched card form.

Functions are referred to in the usual mathematical form (e.g., $cr(vr)$). If the tabulation of a function corresponds to equal intervals in the independent variable, the range of interest can be specified by the first, the last, and the interval between values of the independent variable. In such instances the independent variable (e.g., vr) has an identity of its own and can appear explicitly in equations. Subscripts are nonetheless inherent in a digital machine and the values typed by the computer in response to the min, max, del input (e.g., $MIN = -50$ $MAX = +50$) are the subscripts assigned to the first and last values of the function (using the convention, subscript = vr/del); the user will not usually be concerned with these numbers. If the function values do not correspond to equal intervals in an independent variable, the user may type integers for min and max which are then taken to specify the exact subscript sequence desired.

The results of an operation, such as the evaluation of the arithmetic expression to the right of the = sign, are not automatically output. It is assumed that most results are intermediate and that the user does not want them printed or displayed. However, the simple request

```
print normda(vr)
```

would produce a listing of the calculated function;

```
print normda(vr) -1. 1.
```

would yield only those results for $-1. \leq vr \leq +1.$;

```
print normda(vr) -1.5 2.5 0.025
```

would cause a new array, including interpolated values, to be generated and printed for $-1.5 \leq vr \leq 2.5$ at an interval in vr of 0.025.

The ability to selectively control the output and to specify the currently most meaningful range and interval has reduced the volume of printed output in comparison with batch processing and also facilitates the user's comprehension of his results.

The second step in the data analysis is to generate the theoretical incident intensity distribution, $i(v)$.

```
(i(v) = sqrtf(gamsqs)/(pi*(4.*v**2+gamsqs)))
```

PLEASE PRINT ON THE NEXT LINE MIN,
MAX AND DEL FOR THE VARIABLE V.

```
-1.    1.    .05
```

MIN = -20 MAX = 20

DEC. VALUE OF CONSTANT GAMSQS
PLEASE 0.01

COMMAND PLEASE

In this equation, the system assumes that v is the name of an independent variable in the arithmetic expression, rather than a constant, since it is used as the independent variable of the answer array, $i(v)$. Since no functions appear in the expression, the desired range of v is not defined, and is therefore requested. The particular range used here, $-1.$ to $+1.$, should be adequate to include all the significant values of the peak. If the user had not previously thought much about this point, and if he were at one of the consoles which had display facilities, he would get a quick verification by displaying the resulting function. It is important to note, however, that the system does not depend on the availability of graphical display hardware; printed results are produced in an easily readable form and should always be sufficient, though not necessarily always as convenient.

A graph of the function $i(v)$, which is shown in Fig. 1, could be generated by using a plot request in the following manner:

```
plot
```

PLOT WILL PRODUCE A GRAPH OF THE DE-
SIRED FUNCTION(S).

WHAT FUNCTION(S) WOULD YOU LIKE TO
PLOT. i(v)

SHOULD THE PLOT BE LINEAR, LOG-LOG,
LINEAR-LOG, OR LOG-LINEAR. linear

DO YOU WANT A POINT OR LINE PLOT OF
I(V). line

IF YOU DO NOT WANT ALL OF THE POINTS
OF THE FUNCTION(S) PLOTTED, TYPE THE
RANGE AND/OR INTERVAL IN V TO BE
USED. OTHERWISE GIVE JUST A CARRIAGE
RETURN.

COMMAND PLEASE (The plot would now be on
the display screen)

With the above request we have illustrated the "long form" of a request, in which the user need type only the one word which is the common name for the mathematical procedure or data presentation technique desired. The response of the computer will then be sufficiently explicit, hopefully, that even an inexperienced user will be able to convey the necessary (or optional) specific information. However, such "conversations" may become tedious because after some experience one can anticipate the questions. Therefore optional "short forms" may be used in which some or all of the parameters are presented with the original command. If parameters are given with the command they must usually be presented in the order in which they would have been requested. If only some of the *required* parameters are given, the unspecified ones will be requested. Certain procedures, however, will assume that a particular option is desired if nothing is stated to the contrary. The "short form" has been illustrated previously, in fact, with the various options of the print request.

It may be noted, in the question for $i(v)$, that operational functions may be used (e.g., `sqrtf`), utilizing the names and `f` suffixes which are familiar to FORTRAN users. In addition to those operations which are normally available in programming lan-

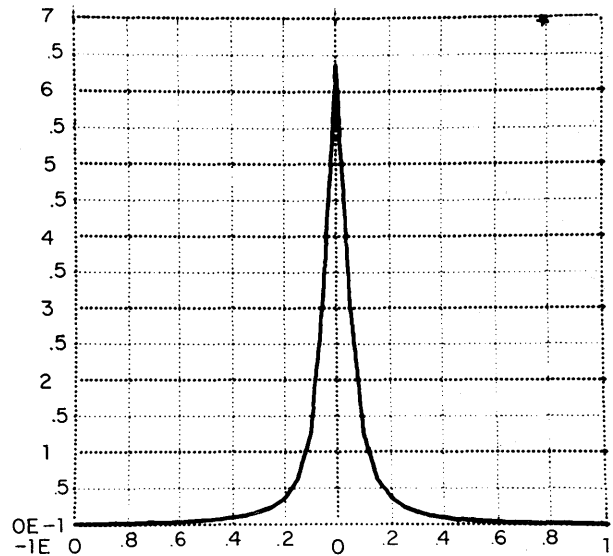


Figure 1. Graph of $i(v)$ versus v generated by the "plot" command, as described in the text. Only the origin is labeled with the appropriate power of 10.

guages (`sinf`, `sinhf`, `asinf`, `sqrtf`, `absf`, `expf`, etc.), MAP includes procedures which operate on all or part of a function simultaneously, such as the summation of all tabulated values (`sumf`), the total integral over the entire range of a function (`intf`) and the derivative (`derif`).

Proceeding with the analysis,

```
(c(vdum)=conb*(qa*gmsqa/(4.*(vdum-va)**2+gmsqa)
```

```
MORE LEFT PARENS THAN RIGHT, CONTINUE OR GIVE CARRIAGE  
RETURN TO CANCEL.
```

```
+qb*gmsqb/(4.*(vdum-vb)**2+gmsqb)+qc*gmsqc/(4.*(vdum-vc)**2+gmsqc)))
```

```
PLEASE PRINT ON THE NEXT LINE MIN, MAX AND DEL FOR THE  
VARIABLE VDUM.
```

```
-5. 5. .05
```

```
MIN = -100 MAX = 100
```

```
DEC. VALUE OF CONSTANT GMSQC PLEASE. .12
```

```
(all other constants not previously defined would be requested in the same manner)
```

```
COMMAND PLEASE
```

```
(abs(vdum)=1-expf(-c(vdum)))
```

```
COMMAND PLEASE
```

A wider range, $-5.$ to $+5.$, is required for the significant values of the absorption factor $\text{abs}(\text{vdum})$, than was required for $i(v)$. Again, since no functions appear in the equation for $c(\text{vdum})$, a range and interval is requested for vdum . If v had been used as the independent variable, the system would have used the previously assigned range, $-1 \leq v \leq +1.$, automatically. The dummy variable, vdum , has been used in order to allow specification of the more extensive range that is required for the abs function and hence the c function.

The calculated spectrum is then obtained by convolving the incident intensity distribution with the absorption factor.

convolve

THIS COMMAND OBTAINS THE INTEGRAL OF THE EXPRESSION $A(X)*B(R-X)*DX$ FOR ALL PERTINENT VALUES OF R .

WHAT IS THE NAME OF THE FUNCTION OF THE TYPE $A(X)$. abs(vdum)

WHAT IS THE NAME OF THE KERNEL FUNCTION. i(v)

NAME OF ANSWER PLEASE. spect(vr)

COMMAND PLEASE

For the convolution procedure, the only requirement on the two input functions is that they each be tabulated at equal intervals in their independent variables. The computer will interpolate (second order) if it is necessary to compensate for an initial inequality of the intervals in the two functions. The resultant function, $\text{spect}(\text{vr})$, will be generated with the interval common to both, or the smaller of the two, and with the augmented range implicit in the convolution. Here, for example, the range of $\text{spect}(\text{vr})$ will extend from $-6.$ to $+6.$ In all the procedures, including equations, the system will handle mixtures of functions with different ranges and intervals and automatically derive results only for the range in which all the functions are defined. When mixed intervals appear, the user will be interrogated to determine whether finer or coarser specifications of the functions are more appropriate to his problem.

Having calculated a first approximation to the fitting function, the user can check the fit using the graphical display. The following is an illustration of a short form of the plot command.

plot normda(vr) points spect(vr) lines $-5.$ $5.$

Unless otherwise specified, the range used for the graph, when two or three functions are displayed, would be sufficient to include all the values of the various functions. Here the user has limited the range to $-5. \leq \text{vr} \leq +5.$, the interval covered by his data. It may be noted that the comparison, shown in Fig. 2, is reasonably good but nonetheless capable of improvement. The user would presumably go through the procedure again, starting at the calculation for $c(\text{vdum})$, after deleting or redefining any parameters which he thought required alteration.

COMMAND SEQUENCES

Since at least two iterations and probably more will be required for most procedures of the type illustrated, it is advantageous if the computer stores particular sequences of commands. This facility is available in MAP, and would normally be used for this problem. After generating the normalized data, $\text{normda}(\text{vr})$, and the incident intensity, $i(v)$, the user need only type the word "create" to define a sequence of commands.

create

TYPE IN COMMANDS, ONE PER LINE. WHEN ALL COMMANDS HAVE BEEN ENTERED, GIVE TWO CARRIAGE RETURNS, EDIT IF NECESSARY, AND GIVE COMMAND 'FILE XXXXXX' (WHERE XXXXXX IS A NAME OF 6 OR FEWER CHARACTERS BY WHICH YOU CAN IDENTIFY YOUR COMMAND SE-

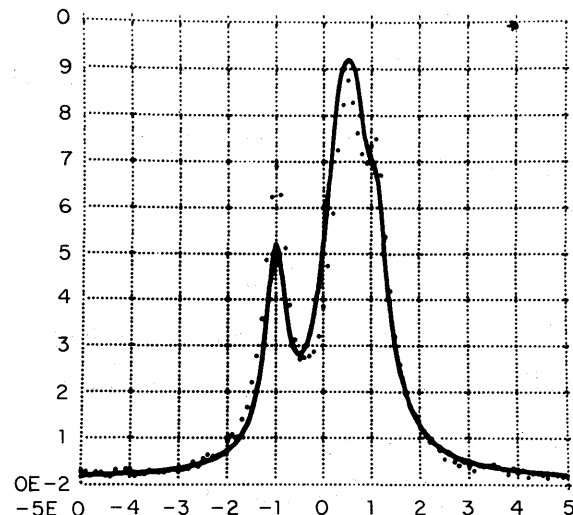


Figure 2. Simultaneous plot of $\text{normda}(\text{vr})$, as points, and the initial calculation of $\text{spect}(\text{vr})$, as a continuous curve.

QUENCE). THE COMMANDS CAN BE EDITED AND PRINTED BY USING THE CONVENTIONS GIVEN IN THE MANUAL.

INPUT:

```
(c(vdum)=conb*(qa*gmsqa/(4.*(vdum—va)**
2+gmsqa)
+qb*gmsqb/(4.*(vdum—vb)**2+gmsqb)
+qc*gmsqc/(4.*(vdum—vc)**2+gmsqc))
(abs(vdum)=1—expf(—c(vdum)))
convolve abs(vdum) i(v) spect(vr)
plot spect(vr) lines normda(vr) points —.5 5.
0.1
```

EDIT:

```
file mossb
```

COMMAND PLEASE

The user has created a “command sequence” and assigned to it the name “mossb.” This list of commands may be edited either before the file request, or at any later time (using the request “edit mossb”). Execution of the entire sequence can be initiated by typing “run mossb.” Unless it is specifically erased, the sequence will remain available for use during subsequent console sessions. During execution each statement in the command sequence will be executed

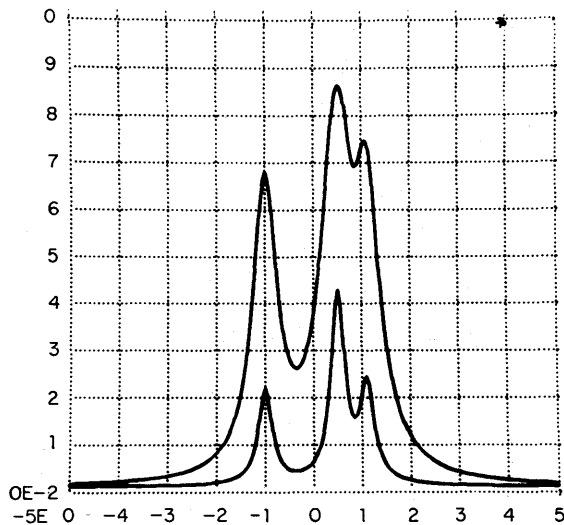


Figure 3. Simultaneous plot of normda(vr), as points, and the final fitting function, spect(vr), as a continuous curve.

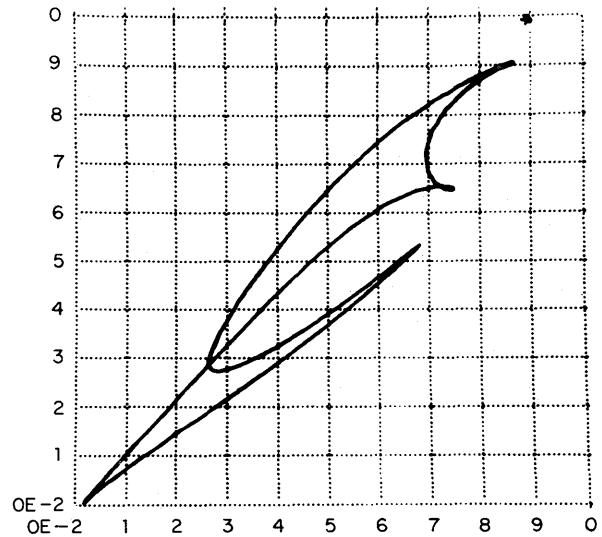


Figure 4. The initial fitting spectrum, spect(vr), plotted versus the normalized data, normda(vr). A diagonal straight line would represent perfect agreement; the three cusps correspond to the three peaks in the spectrum.

just as if it had been typed in at that moment. Having created mossb, the user’s typing could be limited to a series of run mossb’s and the adjustment of parameters between iterations, repeating the sequence until satisfactory agreement was achieved. The complete flexibility of on-line operation is retained, of course, since the user may inject side calculations of any sort between iterations, and may even work on an entirely new problem; the command sequence will remain defined until he deletes it from the system.

It is worth emphasizing that the subsidiary results of command sequences remain available. For example, after finding the best fit the user might want to compare the spectrum to c(vdum), which is more directly related to the physical nature of his specimen.

```
plot spect(vr) c(vdum) —5. +5. 0.1
```

would produce the graph in Fig. 3. Another sort of comparison, which may be useful whenever the result of a transformation or calculation must be compared to an input function, is to plot one function against another:

```
compare spect(vr) normda(vr)
```

presents a plot of normda(vr) versus spect(vr). The graph shown in Fig. 4 was produced when the initial fit was compared to the data. Though they are not

commonly used at present, such graphs provide a wealth of information to the experienced eye.

Unless further calculations are required which involve these particular data and calculated functions, the user would ultimately erase them, but he would probably prefer to retain the command sequence "mossb." If he types "data update," the computer will type out the names of all constants, functions and command sequences that he has defined within the MAP System and will ask him which ones ought to be raised to a more permanent storage level; he should respond "mossb." The subsequent request, "data restore," will erase everything else that has been defined.

We have as yet said nothing about user errors which commonly occur and which lead to serious frustrations when a system is not sufficiently attuned to their likelihood. Insofar as typing mistakes are concerned, MAP makes use of the CTSS erasure facility; typing n quotation marks deletes the n previous characters and a question mark deletes the entire line. All input data and command sequences can be altered immediately after they are typed, or after they have been used by utilizing the "edit" request. Within the MAP procedures themselves, we have attempted to detect all logical inconsistencies and incomplete or impossible requests and, insofar as is possible, to allow the user to correct himself. Often, the user will realize that he has made a mistake when the computer asks him a question that he had not expected; in nearly every case, typing the word "quit" will cause the system to stop the current operation or sequence of operations and revert to "COMMAND PLEASE." If, while waiting for a calculation to be completed, the user realizes that he had sent the system off on a fool's errand, he may interrupt the system by twice depressing the "quit" button on the console.

USER PROGRAMS WITHIN MAP

We have traced the spectrum-fitting problem along those lines which we would expect a student or sporadic user to follow. However, for the researcher who spends his days analyzing Mössbauer spectra, command sequences may not be entirely satisfactory. He may have in mind a more sophisticated analysis technique than is immediately available in MAP, or might prefer to sacrifice some of the ease of obtaining the first solution for the sake of increased efficiency over the long run (when long equations are

involved, the interpretation procedure used in MAP will be less efficient than a compiled program). In order to accommodate such users, the MAP System contains features which facilitate the writing of programs and allow execution of those programs to be intermixed with the basic procedures already available. Suitable programs may be written in any language which can call a MAD subroutine²⁰ (e.g., FAP, MAD, AED, etc.). A MAD program which will exactly duplicate the command sequence illustrated above is shown in Fig. 5.

This MAD program could be written, translated and filed in MAD and BSS form, using the facilities available within the time-sharing system.² The CTSS command "resume map" would then return control to the MAP System. If the program had been assigned the name "pmossb," it could be executed at any time, even in the middle of a command sequence, with the MAP command "execute pmossb." With this particular program, "execute pmossb" would be exactly analogous to "run mossb" using the previously illustrated command sequence. The numerical results would be identical and the computer's

```

INTEGER MIN, MAX, I
DIMENSION C(1000), ABS(1000)
QA=VALUE. ($QA*$)
QB=VALUE. ($QB*$)
QC=VALUE. ($QC*$)
VA=VALUE. ($VA*$)
VB=VALUE. ($VB*$)
VC=VALUE. ($VC*$)
GA=VALUE. ($GMSQA*$)
GB=VALUE. ($GMSQB*$)
GC=VALUE. ($GMSQC*$)
CON=VALUE. ($CONB*$)
EXECUTE RANGE. ($VDUM*$, MIN,MAX,DEL)
THROUGH LOOP, FOR I=MIN, 1,I.G.MAX
V=I*DEL
C(I-MIN)=CON*(QA*GA/(4.*(V-VA).P.2+GA)
1 +QB*GB/(4.*(V-VB).P.2+GB)
2 +QC*GC/(4.*(V-VC).P.2+GC))
LOOP ABS(I-MIN)=1.-EXP.(-C(I-MIN))
VECTOR VALUES ARG1=$ABS(VDUM)*$
EXECUTE OUT. (ARG1,ABS,MIN,MAX,DEL)
VECTOR VALUES ARG2=$ABS(VDUM) I (V) SPECT (VR)*$
EXECUTE CONVOL. (ARG2)
VECTOR VALUES ARG3 $$SPECT (VR) LINES NORMDA (VR)
1 POINTS -5. +5. 0.1*$
EXECUTE PLOT1. (ARG3)
EXECUTE CHNCOM.
END OF PROGRAM

```

Figure 5. Example of the program "pmossb" for use with the "execute" command.

requests for undefined parameter values would also be the same.

Since the MAD program may look more complicated than it is, it may be worthwhile to follow it through in some detail, paying particular attention to the MAP library subroutines which have been used. First of all, one should note that the \$. . . *\$ notation is used in MAD to indicate a BCD specification, i.e., the information within the dollar signs is regarded to be alphanumeric text, with the * indicating to the processing program the end of a specific block of text. The subroutine "value" will obtain the value of a MAP constant from the disk storage. For example,

```
GA = VALUE.($GMSQA*$)
```

obtains the value of the MAP constant named gmsqa and stores the number in the storage unit assigned to the MAD variable ga. If a value is not available, one will be requested in the usual manner.

The subroutine "range" will obtain the range and the interval of a MAP independent variable if that specification already exists. If such values had not been defined previously, they will be requested.

The subroutine "out" will use the values of the specified MAD array to create the named MAP function, with the last three arguments of the calling sequence specifying the range and interval of the function. Since a BCD argument can be included directly as an element in the argument list only if it can be expressed in six or fewer characters (including the final asterisk), the vector values statement immediately preceding the call to "out" has been used to store the necessary argument, \$abs(vdum)*\$. (A MAD "vector values" statement presets and dimensions an array; in this case the array arg1 is created with each word containing successive six character groups of BCD information.) The opposite procedure is accomplished with the subroutine "in," which has the same argument form. The values of the named MAP function are placed in the specified MAD array, and the range and interval of the function determine the values of the last three arguments. If the function had not been defined previously, values would be requested. For both "in" and "out" we use the convention that the first tabulated value in the MAP function, i.e., the one corresponding to min, is the first element of the MAD array, i.e., array(0). It may be noted that this convention is implied in the subscripting used in the equations for the c and abs arrays.

The subroutine "convol" is the exact equivalent of the MAP convolve procedure. The argument should be a BCD list equivalent to the information that would be requested by the MAP procedure, or which could be typed, in whole or in part, on the command line. In this example the program has been preset with the "short form," all the parameters having been included in the vector arg2. At the opposite extreme, the following calling sequence is possible:

```
EXECUTE CONVO1. ($*$)
```

In this call no parameters have been included in the argument list and the execution will be equivalent to typing just "convolve" as a MAP request. All of the MAP procedures have their counterpart subroutines, the names being formed from the first five letters of the procedure name followed by a 1 suffix. In addition to the print, plot and convolve already mentioned, such analyses as integration, Fourier transform, least square fitting, differentiation, interpolation and others are presently available.

The subroutine "chncom" returns control to the MAP System. Generally the next response would be a "COMMAND PLEASE," but if the "execute" had occurred in a command sequence, the analysis would simply continue in its prescribed course.

We have, in this sample program, purposely preset all the parameter names (e.g., the MAP names qa, qb, etc.) so that it would be exactly analogous to the previously illustrated command sequence. A user might find it advantageous, however, if some or all of the parameter names were not specified until execution time. Of course, when MAP procedure subroutines are involved, any parameter names which are not required elsewhere in the program may be left unspecified by giving an empty argument list in the calling sequence. A more general procedure is available, however, in a subroutine called "setup" which will pick up the parameter names presented at execution time on the command line, following "execute pmosb." In the foregoing MAD program the MAD variable con was defined to be equal to the value of the particular MAP constant conb, with the statement con = value.(\$conb*\$). If it were desirable to leave the MAP parameter unspecified until execution time, that statement could have been replaced by the following sequence of statements:

```
EXECUTE SETUP. (NAME(0),1)
NAME(1) = $*$
CON = VALUE. (NAME)
```

At execution, the computer would then look for one additional word on the command line, and would take that word to be the name of the MAP variable whose value should be used for con. As many as 13 parameter names may be supplied at execution time in this fashion; all of them would be picked up by a single execution of "setup," with the second argument in the call to "setup" indicating the number of parameters expected (if the specified number are not supplied, the computer will give the user an opportunity to retype the list of names).

Additional programming subroutines, also automatically available in the MAP library, are designed to further assist the programmer in maintaining contact with the other parts of the MAP system and in achieving a high level of sophistication in the transmission of information in both directions through the console. Since it is easy to convert a program written for the "execute" command into an ordinary MAP command, expansion of the system into particular regions of interest is readily accomplished.

ACCURACY, COMPLEXITY AND TIME

The majority of prospective users of a system such as MAP are relatively naive on the subject of accuracy in numerical analysis. It would therefore be self-defeating to require each user to determine the most appropriate order of approximation to be used for each specific problem (though this might be appropriate in a course on numerical analysis). Rather we have decided that, at worst, each procedure should be (nearly) exact if three point fitting is adequate. Thus the user need only remember that all experimental data and calculated input functions, with the exception of those intended for least square fitting, should be tabulated such that a parabolic fit over three successive points specifies the function as exactly as is meaningful or required. The various procedures (e.g., Simpson's integration, five point differentiation, quadratic interpolation, double precision matrix inversion, etc.) should not lose any of the precision inherent in the input. Nothing prevents the more knowledgeable user, of course, from compiling his own approximation to a given procedure, which he can then use as freely as any of the MAP procedures.

Since MAP is designed to be used by persons who are not expert in numerical analysis, it is necessary that some attempt be made to prevent users from

ascribing an unreasonable degree of accuracy to their results. The proper interpretation of computer output will become increasingly important as more people use computers without understanding the details of the algorithms employed. Although we do not have a solution to the problem in general, we have attempted to provide error messages in all cases where the mathematical operation is not meaningful (e.g., whenever a requested integration range is greater than the range of the integrand) or liable to give grossly inaccurate results (e.g., whenever too few points are available in a function to perform a valid interpolation). In the case of the least square analysis an estimate of probable errors, based on the size of the diagonal elements in the inverted matrix and the agreement between the fitted curve and the input data, is printed. However, for many procedures there are no general methods for estimating the errors. In the last analysis, all users of computers (including those who write their own programs) must be convinced of the necessity of considering all results with a critical eye, and for testing new procedures with test cases or by circular calculation wherever possible. Since all functions generated by MAP are saved and easily available, the user is readily able to examine intermediate results for test purposes.

It ought to be obvious that complex problems (or even simple problems) which require a large amount of machine time between interactions are not particularly suitable for on-line solution. When the required block of machine time is many times larger than the quantum of time allotted to each waiting user, the elapsed time becomes excessive unless one is doing other work simultaneously. Therefore, in addition to making recommendations along these lines in the user's manual, we have attempted to force certain limitations where they seem appropriate. For example, the least square fitting procedure will not accept a problem involving more than 100 data points and 5 fitting functions, though the program which is used is capable of handling up to 1000 points and 20 unknowns. The prospect of on-line initiated batch processing shows promise of being a useful compromise in many instances. The few necessary programs will be written, therefore, so that jobs can be initiated in MAP (retaining all possible user-machine interaction) and the required calculations performed later, utilizing the on-line initiated batch processing facility that is being developed for CTSS.

The question of elapsed time at the console is one of the most critical tests of the feasibility of an on-line system. It is also a difficult test to apply since different users have different states of nervousness while waiting for output and since the waiting time faced by any given user depends on the current efficiency and the load on the overall system. On the basis of two years' experience we have been able to ascertain, however, that whenever service is generally considered to be good by most users of CTSS it is perfectly adequate for MAP. We feel now, for example, that when the maximum number of users (upwards of 30) are logged in, the system response tends to become too slow; with 20 users the computer and the user are roughly equally matched; with 10 or fewer consoles logged in, the user's nervousness may have a reversed source, i.e., the computer may seem to be pushing him.

THE TERMINAL PROBLEM

At present most of the consoles available to the CTSS system are simply teletypewriters, and the time spent at the terminal depends significantly on the volume of input data and printed output. The graphical figures in this paper were generated on the MIT Electronics Systems Laboratory display terminal,²¹ which provides two users simultaneously with a CRT display, a light-pen, and analog devices for input and manipulation of displayed patterns. This facility is expensive and requires a direct data channel to the computer; it is therefore not a feasible terminal to provide to many users. The Project MAC display facilities have been augmented with a number of storage oscilloscopes which will shortly be available at various remote consoles, utilizing switchable telephone lines.

At the present time, we may make some suggestions as to the desirable features for remote terminals to be used with a system such as MAP: (1) typewriter input and rapid output; (2) rapid and locally maintained display, in a moderately lighted room, of curves, points and formatable text, including the option—or alternative—of hard copy; (3) graphical input (curve tracing and free hand); (4) a device for automatic input of large blocks of previously recorded data (probably punched paper tape). More sophisticated units, such as those required for display of three-dimensional surfaces, would be desirable, but could be available only at a few central locations.

APPLICATIONS IN RESEARCH AND TEACHING

In the Introduction we mentioned that 30 to 50 mathematical procedures would be desirable in a system of general usefulness. Actually, we have been using MAP effectively in its present and lesser states of development even though only 14 procedures have been implemented so far. It has been of value in analyzing data from a wide variety of experiments and in various theoretical calculations.

The system has also been used in the teaching of three lecture courses, two graduate and one undergraduate, all concerned with the physics of solids, and in a laboratory course in solid state physics. In these trial subjects we have used MAP to allow the students, working in pairs, to do computational problems related to the experiments or to the lecture material. Through a variety of problems we have tried to use the computer to deepen their understanding of the basic subjects, rather than as a separate topic itself. In more definite terms, we may list four specific points:

1. Study of the form of complex analytical and numerical solutions.
2. Study of the effect of variation of the parameters in a physical situation.
3. Introduction to and experience with some of the mathematical procedures important to the subject.
4. Development of a sense of familiarity with the subject through working with the equations and numbers in which it is expressed.

Since the majority of the students had no previous computer experience it would have been impossible to use the computer at all, except for demonstrative purposes, without a system such as MAP. Even an experienced programmer would have required the equivalent of many days of class time to obtain a working program for some of the problems which were solved. In practice we have found that the students required only a couple of hours of attention during their first sessions at the console, and thereafter were able to work quite independently.

The fact that a relatively small number of procedures is yet available is more seriously felt in teaching than in research applications. Even if nothing more were available, for example, than the equation interpreter and the Fourier transform pro-

cedure, many experimenters would find the system useful. Even a single course, however, tends to require a greater generality. Thus, though we have been able to treat such class problems as the forced motion of a particle in an arbitrary potential, the relationship between the real and momentum space wave functions, the relationships between expectation values, charge density and scattering factor, and the effect of various parameter combinations in the Kronig-Penney model for periodic potentials, we have had to avoid any topics in the courses which would have required matrix operations, differential equations and functions of more than one variable. These latter procedures, plus the solution of integral equations, we intend to install in the near future.

It should also be mentioned that all of the research and teaching applications were actually implemented without any display facilities. As has been previously discussed, such facilities are now available (and are being further extended and tested) and will be a great advantage to subsequent users.

In closing we should mention that insofar as teaching is concerned, we have barely touched the surface. One can readily visualize, for example, the inclusion of particular problem-solving and teaching programs. As examples of such programs we might consider the nuclear reactor design programs⁶ mentioned in the introduction and the "Socratic" teaching system which has been discussed by J. A. Swets and W. Feurzeig.¹⁸ In each case the combination of general mathematical procedures with specific programs aimed at particular subjects should provide powerful tools which will have a profound influence on teaching concepts.

ACKNOWLEDGMENTS

The MAP System would not have been possible without the existence of the MIT Compatible Time-Sharing System which is due primarily to the staffs of the MIT Computation Center and Project MAC. Virtually all of the applications of the system, in both research and teaching, were accomplished at the Center. We are indebted to the authors of various programs which were utilized in the system; in particular, we would like to thank Dr. Thomas G. Stockham, Jr., of the MIT Lincoln Laboratory for his assistance in implementing the graphical displays. It is a pleasure, also, to thank Professor B. L. Averbach, of the MIT Metallurgy Department, for his encouragement in the development of MAP, particularly in its application to teaching.

REFERENCES

1. R. M. Fano, "The MAC System: The Computer Utility Approach," *IEEE Spectrum*, vol. 2, pp. 56-64 (Jan. 1965).
2. P. A. Crisman (Ed.), *The Compatible Time-Sharing System: A Programmer's Guide*, 2d ed., MIT Press, Cambridge, Mass., 1965.
3. D. Roos, "An Integrated Computer System for Engineering Problem Solving," *Proceedings of the Fall Joint Computer Conference*, vol. 27, Spartan Books, Washington, D.C., 1965.
4. I. R. Whiteman, "New Computer Languages," *Int. Sci. and Tech.*, Apr. 1966, pp. 62-68.
5. M. L. Dertouzos and J. F. Reintjes, "Computer-Aided Electronic Circuit Design," Project MAC Progress Report II, July 1964-65, MIT, 1966.
6. K. Hansen and I. C. Pyle, "TREC: A Time-Sharing Reactor Codes System," *ANS Trans.*, vol. 7, p. 2 (Nov. 1964).
7. M. Greenberger et al, "On-Line Computation And Simulation: The OPS-3 System," MIT Press, Cambridge, Mass., 1965.
8. R. Kaplow, J. W. Brackett and S. Strong, "MAP, A System for On-line Mathematical Analysis: Description of the Language and User Manual," Technical Report MAC-TR-24, MIT, 1966.
9. J. C. Shaw, "JOSS: A Designer's View of an Experimental On-Line Computing System," *Proceedings of the Fall Joint Computer Conference*, vol. 26, Spartan Books, Washington, D.C. 1964, pp. 455-64.
10. G. J. Culler and R. W. Huff, "Solution of Nonlinear Integral Equations Using On-Line Computer Control," *Proceedings of The Spring Joint Computer Conference*, vol. 21, Spartan Books, Washington, D.C., 1962, pp. 129-38.
11. —, and B. D. Fried, "An On-Line Computing Center for Scientific Problems," Thompson Ramo Wooldridge Computer Division Report (now Bunker-Ramo Corp.), Canoga Park, Calif., June 1963.
12. —, "The TRW Two-Station, On-Line Scientific Computer: General Description," *Computer Augmentation of Human Reasoning*, Spartan Books, Washington, D.C., 1965.
13. J. Weizenbaum, "Eliza A Computer Program for the Study of Natural Language Communication Between Man and Machine," *Comm. ACM*, vol. 9, pp. 36-45 (Jan. 1966).
14. D. T. Ross and C. G. Feldman, "Computer-Aided Design," Project MAC Progress Report II, July 1964-65, MIT, 1966.
15. J. Katzenelson, "AED-NET: Simulator for Nonlinear Networks," *Proceedings of the IEEE* (to be published in 1966).

16. M. M. Kessler, "The M.I.T. Technical Information Project," *Physics Today*, Mar. 1965, p.28.

17. C. Levinthal, "Molecular Model Building by Computer," *Sci. Am.*, June 1966, p. 42.

18. J. A. Swets and W. Feurzeig, "Computer-Aided Instruction," *Science*, vol. 150, p. 572 (Oct. 29, 1965).

19. J. C. R. Licklider, "Man-Computer Partnership," *Int. Sci. and Tech.*, May 1965, p. 18.

20. *The Michigan Algorithm Decoder (MAD) Manual*, University of Michigan Computation Center, Ann Arbor, Mich., Oct. 1965 and previous eds.

21. J. E. Ward, "Display Systems Research," Project MAC Report: Progress to July 1964, MIT, 1965.

THE CHECK PAYMENT AND RECONCILIATION PROGRAM OF THE U. S. TREASURY: PRESENT STATUS AND FUTURE PROSPECTS

George F. Stickney

Department of the Treasury, Washington, D. C.

INTRODUCTION

On October 14, 1955, the Secretary of the Treasury, the Comptroller General of the United States, and the Director of the Bureau of the Budget, jointly announced the adoption of new procedures involving the use of electronic data processing equipment, for the "payment" and "reconciliation" of the 350 million checks drawn annually by over 2,300 individual government disbursing officers against the Treasurer of the United States. They announced that they expected these new procedures to save the government \$1.75 million in administrative costs annually and that further decreased costs of about \$500,000 could result in the Federal Reserve Banks.

Actually, adoption of the new system resulted in an annual savings of about \$4 million and involved a reduction of about 800 employees. In testifying on the appropriation for the Office of the Treasurer of the United States before the Subcommittee of the Committee on Appropriations, House of Representatives, in March 1966, Secretary Fowler said:

Fiscal year 1967 will mark the 10th anniversary of the use of electronic data processing equipment by that office in support of its check handling activities. By the end of this 10-year period, the use of that equipment will have saved the Government over \$30 million. Fiscal year 1967 will also mark the full recovery of the capital investment expended in prior years for the purchase of electronic equipment. Savings resulting from owner-

ship rather than leasing will then equal the cost of the equipment.

When this equipment was installed in fiscal year 1957, the annual check volume was 363 million; in 1967 a check volume of 522 million is expected. The same equipment used to handle the 522 million checks will also be used to process an estimated 210 million postal money orders for the Post Office Department on a reimbursable basis.

There is attached as Appendix B a synopsis and cost analysis of the EDP program in the Office of the Treasurer, U. S. This information was furnished the Appropriations Subcommittee of the House of Representatives on March 3, 1966, at its request.

DEFINITION OF PAYMENT AND RECONCILIATION

"Payment" and "reconciliation" of checks involve control processes with which almost everyone is familiar. Everyone who has a checking account at a bank understands that the bank "pays" the checks drawn by him by charging them against his account. He knows further that in this "paying" process the bank must set up controls to avoid, among other things "paying" checks: (1) which do not contain an authorized signature, which contain evidence of alteration, or are otherwise improperly drawn; (2) when the bank has previously been supplied with a "stop-payment" notice; and (3) when there is an insufficient balance in the drawer's account.

The holder of the checking account is also quite familiar with the operation of "reconciliation" of the checks drawn by him with the checks "paid" by the bank and returned to him with his statements of account. He knows he must effect a proof of the paid checks with his issue records and that he must develop the amount of outstanding checks in order to reconcile his balance with that shown by the bank statement.

The processes of "payment" and "reconciliation" of government checks are basically the same as these simple processes. Therefore, a study of the basic features of the program for use of electronic data processing in this area provides a rather unique opportunity of exploring the implications of these advanced techniques in terms of application to simple and widely understood control processes. Such a study should bring out the fact that even the simplest of procedures must be completely "re-thought" in terms of their objectives, as distinguished from existing routines, to provide a basis for application of electronic data processing. Of further interest will be the great amount of research and study which is involved in this "re-thinking" process and in the development and installation of the electronic procedures to meet established objectives. The organizational impact which results from the adaptation of these advanced techniques to even these simple processes will be another matter of special interest. The effect of the installation of these new procedures on traditional concepts of auditing and internal control will also be discussed in this paper. Finally, there is discussed also very briefly the future prospects of a checkless-no-money-economy.

GENERAL OUTLINE OF PREVIOUS PROCEDURES

There is no fundamental difference between the functions of "payment" and "reconciliation" of checks in the Federal government and commercial practice. It seems necessary, however, to provide a general outline of the areas of responsibility involved in the government's disbursing processes so that such similarity can be recognized in terms of the organizational structure of the Federal government.

The outline, which is included as Appendix A, is intended only to provide general background with respect to the basic processes, and related alignments of responsibility. It does not deal with many different procedures which cover various special problems in this general area.

SUMMARY OF PRESENT SYSTEM

The government now issues an average of two million checks daily. The checks are payable at Washington, D. C. The payment and reconciliation functions are performed through the use of an electronic system composed of one large transistorized computer, a smaller auxiliary-type computer, and a battery of card-to-tape (specially designed) converters, which are operated off line.

Each disbursing officer is required to furnish either listings or reels of magnetic tape of all checks written. These listings or reels of tape, which contain the detailed information on each check plus certain block totals, are submitted at least monthly directly to the Treasurer of the United States and subsequently are used for reconciliation.

Following encashment of a check by the payee, it is deposited sooner or later in a commercial bank. The bank will honor the check after proper examination and then will apply to its cognizant Federal Reserve Bank for the reimbursement, which usually takes the form of a credit to the bank's reserve account. The Federal Reserve Bank then applies to the Treasurer of the United States for reimbursement of the amount which it has credited to the commercial bank. When the Treasurer has electronically examined the check to determine that it bears an authorized disbursing officer's symbol and serial number and that there is not a stop-payment notice against it, the check is considered "paid." Checks are received in batches of about 1,000 checks, accompanied by detailed listings.

Card-to-tape conversion:

The first step in machine processing consists of the following operations:

- a. A front-end audit is done on each batch.
- b. Double punch, blank column and other error checks are separated.
- c. A record on magnetic tape is written for each accepted check.
- d. A locator number is printed on each check, which is also incorporated into the tape record.

The purpose of the front-end audit is to establish that the total dollar amount of the checks in each batch corresponds to the charges in the transmittal letter. Imperfectly punched checks are not acceptable

to the system. The converter will route these checks into a separate pocket and not write them on tape. Later on, replacement cards are key-punched for such items, after which they are re-entered.

Checks arrive in random sequence and a batch may contain checks from many disbursing offices. It is necessary that the tape records be sorted in sequence by disbursing officer's symbol and serial number which eliminates the necessity of physically sorting the checks. It is essential, however, to maintain physical access to all checks so that any check can be located and examined for signature, endorsement, etc., upon demand. The converter prints a consecutive "locator" number on the face of each check and simultaneously writes the same number as part of the tape record. The checks are then physically filed in locator number sequence having been handled only once. When the tape records are sorted by disbursing officer's symbol and serial number, the locator number for each check is carried along. Thus, when it is necessary to make a physical examination of a check, the check record on tape can be easily located in its logical sequence by disbursing officer's symbol and serial number and it will show the locator number which will pinpoint the location of the check in the file.

Card-to-tape converters are also used, incidentally, to enter all the other types of transactions, such as issue information and stop payment notices, that are required by the system. Each item bears a transaction code for the computer to use in identifying the type of record.

Computer functions:

After operations are completed on the converter, the check records are ready for processing on the computer. The following major steps are essential in accomplishing the three objectives of payment and stop payment, reconciliation, and check status reporting:

- a. Balance and Condense
- b. Sort
- c. Pay and Stop Payment
- d. Detailed File Maintenance
- e. Reconciliation

Balance and Condense is a preparatory run which normally involves processing a group of 14 to 18 converter reels. Further "grouping" is accomplished during this run, which is designed to reduce the amount of tape handled without reducing the amount of information. There is still process time left over,

and this time is used to do some initial sorting, in preparation for the next run.

Sort is a run which completes the operation of placing the records in sequence by disbursing officer's symbol and serial number. Normally, the Balance and Condense Run and the Sort Run are performed twice daily, with the number of transactions processed ranging between 1.6 and 2 million.

Pay and Stop Payment is the daily ledger maintenance run. A master file is updated which contains a history record for each disbursing officer, showing the disbursing officer's symbol and authorized range of check numbers. The master file also contains all stop-payment notices which are active. Checks from the Sort run are matched against this master file and are either paid, intercepted by a stop payment, or rejected for unauthorized disbursing officer's symbol or range of serial numbers. Paid checks are written out on another tape for use in subsequent runs. Unaccepted checks are written out on a separate tape for analysis. When a stop payment has intercepted a check, the event is recorded on a "print" tape. New stop payments enter the system in the same manner as checks and are placed in proper sequence on the master file. They are also sent forward with the accepted checks to search the detail file in order to determine whether the check was paid prior to receipt of the stop payment.

During the Pay and Stop Pay run, the computer starts to collect and organize information for use in controlling reconciliation. The decision of when and how to reconcile is a fairly complicated one, and a good decision invariably involves a compromise. Treasury has adopted the management by exception technique of reconciling by blocks of checks, and only examining individual checks or issue records when the blocks fail to reconcile. This technique has proven to be a highly efficient and economical method of audit control. In order to use this technique, however, it is necessary to strike a balance between two opposing forces. On one hand, it is highly desirable to reconcile a block as soon as possible, in order to detect error conditions early and to minimize the amount of information which must be carried in float. On the other hand, it is desirable to postpone reconciliation of a block to allow time for the individual checks to arrive. Otherwise, the number of checks which are still outstanding at the time of reconciliation would be unreasonably large and the system would defeat itself. A great deal of thought and statistical analysis have been devoted to this question.

In general, the factors governing reconciliation are:

- a. The elapsed time since the first check was filed in the block.
- b. The number of checks in the block which have been received.
- c. The relationship between total payments and total issues.

Even though reconciliation is performed on a block level, it is still necessary to maintain a complete record of individual checks, so that they may be located and examined in the event that a block fails to reconcile. During the first month or so following the issuance of a block of checks, the rate of arrival is quite high, and it is logical to maintain the detailed records right in the system. Sooner or later, a point is reached where most of the checks are in and it may take months or even years for the remaining few to get cashed and presented for payment. At this point, it makes more sense to remove from the system and print the records of the checks which have arrived, while maintaining in the system records of only those few checks which are still outstanding, and a skeleton record which may be used to identify the listings upon which the checks were printed.

An edit tape is prepared during the Pay and Stop Pay run which is later used in conjunction with an auxiliary computer/printer system to produce various accounting reports reflecting the scope of the day's business.

Detail File Maintenance covers one-fifth of the total file on a daily basis. As previously stated, checks which are found to be acceptable update the Pay and Stop Pay Master File and are recorded on an output transaction file which is divided into five segments. The reason for performing this segmentation is that the Treasurer's Detail File Maintenance operations are cycled in such a way that one-fifth of the files are serviced each working day, with the entire file being serviced every five days.

Corresponding output transaction file segments of paid checks and related transactions from each of the five preceding working days are merged into a single sequence, in order by disbursing officer's account and check serial number. This combined segment, representing a week's accumulation of paid checks for the accounts encompassed in one particular fifth of the file, is now ready for posting to the corresponding segment of the detail file. Each segment of the detail file is quite large, in that it contains data for about 40 to 50 million checks which have been paid but have

not yet been reconciled, at any given time. Since the basic purpose of the file is to show which checks have arrived and where they have been physically filed, the information carried for each check is the locator number which was assigned back in the card-to-tape conversion operation. Each check which has arrived will have a locator number stored in this file. By the same token, the absence of a locator number for any given check number is proof that the check itself has not yet arrived.

In order to conserve space on tape, a technique was developed for storing all this information without having to record the serial numbers for each check. For each group or block of 100 checks, the serial number of the block alone will be shown; the locator numbers for the individual checks are placed in predetermined positions during the posting process in such a way that when the record is eventually printed on a matrix or grid type of form paper, the locator number for any given check may readily be determined by reference to the matrix coordinates. Before the locator number of any check is stored or posted to the detail file, however, the predetermined position is examined to see if a locator number is already there, in which case a duplicate check condition has been discovered, and the computer will branch into another routine to initiate an investigation.

The printout, which is eventually made for all blocks, also contains line and column totals of amounts which will assist a clerk in tracking down out-of-balance conditions.

The technique which has been described considerably reduces the size of the file since the serial number for each check can be deduced and need not be recorded. The ratio of condensation is 30 to 10.1.

Reconciliation is performed for each block of checks. When any given block of checks is "flushed out" of the detail file (for later printing on form paper), each locator number "pocket" is examined and an "outstanding check" serial number is generated for each position which is blank. These outstanding serial numbers, along with the block totals, are recorded on an output reel of magnetic tape which is used later to update an Outstanding Master file. In subsequent weeks, when any of these outstanding checks arrive, they are processed through the Pay and Stop Pay run and the Detail File Maintenance run. They pass through the latter run and are written on an output tape which will be run against the file of outstanding numbers. There, they

will match up with previous outstanding serial numbers, thereby reducing the number of items carried in the system as "outstanding."

When the age of the block indicates that reconciliation is imminent, the computer is programmed to cause the remaining outstanding check serial numbers, in the case of non-tape accounts, to be punched out on mark-sense type punch cards. These cards are routed to a reconciliation clerk who has on file the original issue list submitted by the disbursing officer. This issue list contains the check serial number and amount for each check plus an amount total for each block. Each punch card is marked with the appropriate amount and this amount is subsequently automatically punched into the card. Such cards, now completed as to outstanding amounts, are converted to magnetic tape and are then reintroduced into the system. (The preceding steps are unnecessary of course for the "tape-accounts," as the amounts of outstanding items are preserved on tape as furnished by disbursing officer.) This technique has not only made it possible to reconcile each block of work in the system automatically, reporting out for investigation only those few blocks found to be out of balance, it also furnishes the advantage of future protection by immediately disclosing any straggler check received

that has been altered or raised in amount and which might otherwise remain undetected until such time as the block became overpaid.

During the entire process, amount totals at various levels and other accounting controls are continuously generated and compared to assure complete audit protection of the system.

SIGNIFICANT TECHNIQUES

In retrospect, the new system embodied some significant techniques which came about by "re-thinking" some of the procedures previously followed in terms of their objectives.

File locator number:

Probably the most significant technique in the new system is the file locator number shown in Fig. 1. This is a progressive number and is printed on each paid check and added to tape record by the card-to-tape converter. It may be viewed as the second serial number. This technique permits us to handle the checks only once and eliminates the necessity for sorting the documents by symbol and serial number. Under previous procedures they were handled 15 to

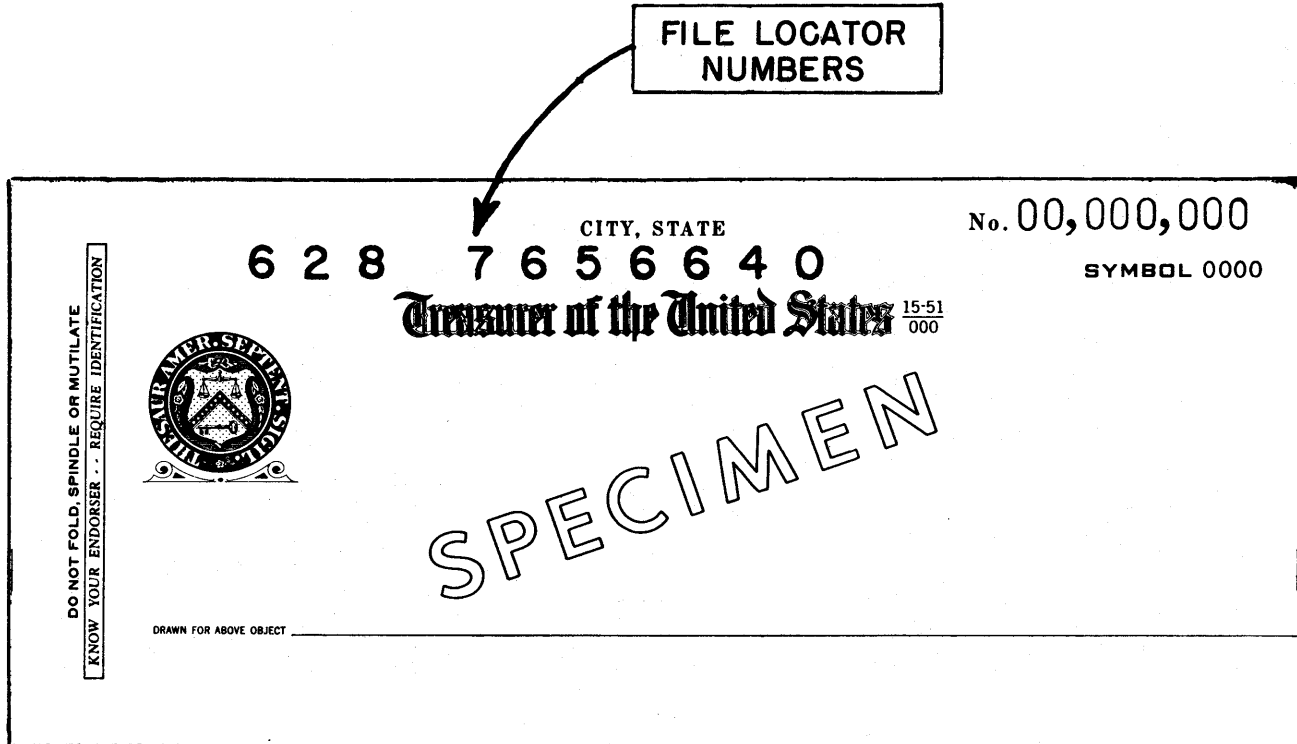


Figure 1. File locator numbers.

20 times during the payment and reconciliation functions.

Elimination of sorting checks:

Historically speaking, it was always necessary under previous procedure to physically sort paid checks at least in order by disbursing officer to obtain the total amount of payments to be posted to the drawer's account. In view of the large volume of government checks, particularly in some disbursing accounts, it was necessary to further arrange the paid checks for each drawer in serial number sequence. The arrangement in serial number sequence was required to facilitate reconciliation of paid checks with issue records and to provide a basis to locate paid checks subject to claim of non-receipt by the payee.

Each working day about 2,000 claims (stop payments) are received which require an examination of the subject check (if previously paid) to determine whether an investigation of forgery is to be made by the Secret Service.

Again the use of the file locator number eliminated the necessity for sorting paid checks in sequence by drawer's account and check serial number which represents 12 digits of numerical information for

each record. It is interesting to note that at the time of conversion in 1956 there were more than 150 numerical card sorters used for this purpose.

Construction of master file:

The master file of paid checks on tape has been designed to record each paid check in a matrix format which is illustrated by a sample print out on Fig. 2. The "matrix" permits reduction of about 66 percent of the numerical digits comprising each paid check record obtained from the card-to-tape conversion. More specifically, originally each paid check record is represented by 30 digits of numerical information. When filed in the (matrix) master file of paid checks each record requires an average of 10 plus numerical digits.

One will observe that what has been accomplished is the elimination of repeating the (1) disbursing officer's account symbol, (2) check serial number, and (3) individual amount. Again, the file locator number, which is recorded in prepositioned spaces on the master tape depending on the check serial number, is the factor which permits this substantial reduction in the number of required digits of data.

In brief, the file locator number technique is un-

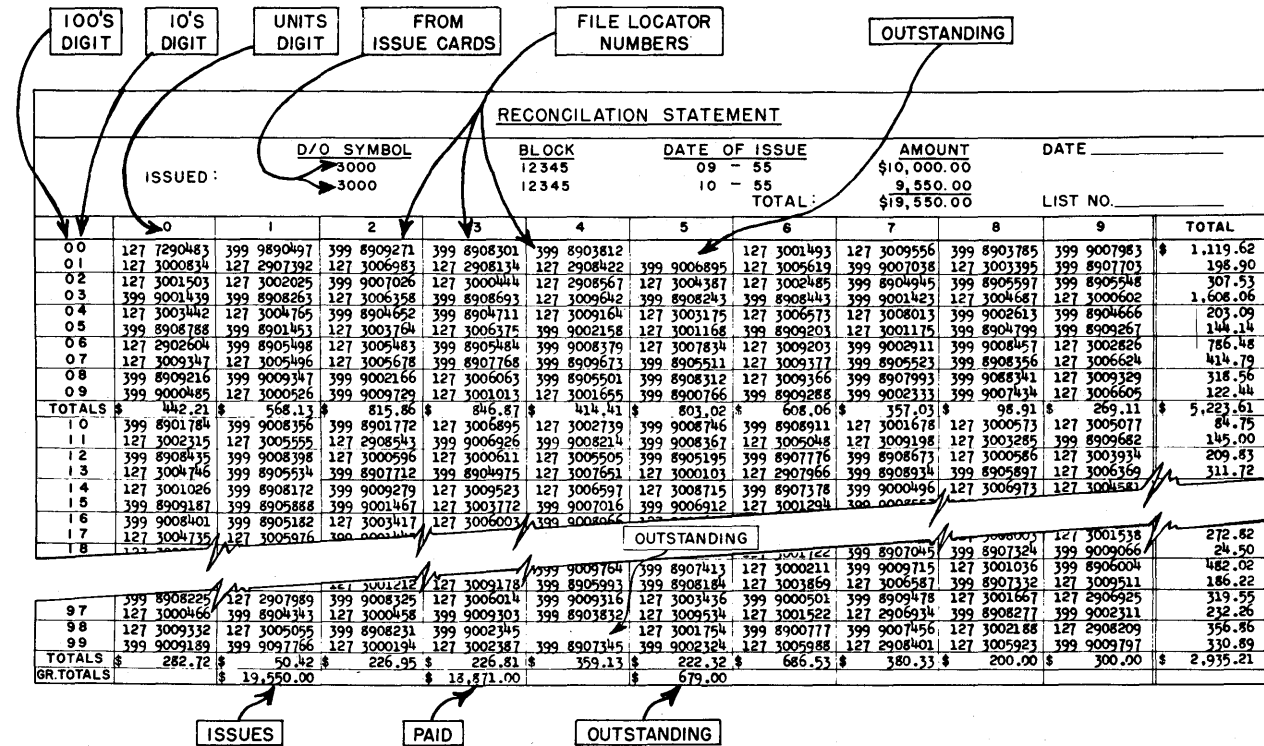


Figure 2. Printout of master file matrix.

doubtedly the most significant factor in the entire process and its use has been the major factor in realizing very substantial economies in operations.

EFFECT ON EMPLOYMENT

Probably no single change in procedure involving a simple repetitive operation ever had as large an impact on organization and employment in the Federal government as the introduction of an electronic data processing system for the payment and reconciliation of checks. Once the recommendation was approved to use the electronic data processing method, our problems in personnel became very real.

More than 50 percent (net) of personnel required under previous procedures was eliminated. The new function required testing, selecting, and training of employees for the new system. In order to place the personnel, whose services were not required under the new plan of operation, a program of testing and retraining employees for other types of employment was carried out over a period of more than a year.

Among many questions which faced us, were—How were we to obtain qualified programmers and console operators? Where would we relocate the majority of our people whose positions were being abolished by the installation of the electronic data processing equipment? In the event we could not produce sufficient qualified programmers, to what sources would we go? To what extent should we go in attempting to train some of our people in other lines of work? What was to be an adequate rate of pay for programmers and operators? You must bear in mind that until this time there had been no positive reason for us to be concerned with the potential of our employees in other than non-technical types of work. A great many of our people had been with us since World War I and over the years had advanced to higher graded clerical and administrative types of positions which did not require, in most instances, formal education beyond high school nor aptitude for scientific type positions. Briefly, they composed the nucleus of dependable public servants who were dedicated to the performance of their jobs in an efficient manner.

Our first order of business was to determine who of our employees were considered potential programmers and console operators. By memorandum, an invitation was made to employees in the Check Payment Division of the Treasurer's Office and the

Check Reconciliation Branch of the General Accounting Office to qualify for training as programmers and console operators. A battery of aptitude tests which consisted of arithmetic reasoning, association of symbols, etc., was administered to 82 employees of the Check Payment Division and 130 employees of the General Accounting Office. Of this group, 24 (8 from Treasurer's Office, 16 from General Accounting Office) were selected for training. Since we were unable to obtain a sufficient number of employees to send to Programming School from the immediate areas affected by the conversion program, an invitation was extended to employees in other areas of the Treasurer's Office and the General Accounting Office to take the aptitude test. 95 additional employees of the Treasurer's Office took the test, 7 being selected for training. 37 employees of the General Accounting Office took the aptitude test and 10 were selected for training. At this point, supervisory evaluations were obtained on each employee who passed the aptitude test which developed such information as their dependability, attitude, ability to work under constant pressure, ability to accept frequent changes in assignments, etc.

Personal interviews were held between applicants and operating officials during which the applicants were apprised of the difficulties under which they would work, the rigid deadlines, strenuous training sessions, prolonged and indefinite periods of overtime, etc., so that those who were inclined would have an opportunity to withdraw from competition. Arrangements for the training were made with the manufacturer of the data processing equipment to give a four-to-six week intensive course in the fundamentals of programming. Two identical courses were given to the selected employees, 20 employees attending the first course and 21 the second. Of the 41 participants, 18 made acceptable marks. Weekly progress reports were furnished by the instructors indicating those employees who should be withdrawn because of an inability to grasp the fundamentals of programming, those employees who should be continued in the programming field, and those who would better serve in the capacity of console operators. Upon completion of their intensive training the successful candidates were immediately assigned to develop phases of basic programs including the introduction of their programs into the electronic data processing system so that as many weaknesses as possible would be eliminated before the conversion was effected.

During the period when our programmers and console operators were being trained by the manufacturer, we were concerned with the problem of establishing the organization, developing job descriptions and classifying the positions, which in effect determines the salaries to be paid for specific duties. Since the Treasurer's Office was a pioneer in this field in Government, we actually had very little to work with in the way of precedents for determining salaries. For the benefit of those not familiar with personnel regulations pertaining to Federal Government, it should be explained that in most series or categories such as accountants, stenographers, tabulating equipment operators, claims examiners, etc., guide lines known as Classification Standards are promulgated by the Civil Service Commission. These standards which define responsibilities of work at each grade level are used as a means of evaluating the duties of a particular position under consideration in order to arrive at a proper grade classification.

At the time we were developing the job descriptions for our programmers and console operators there were no Class Standards for the series to which we would allocate these positions. This greatly increased the problem because we were entering the area about which we knew relatively little and, therefore, were hampered in our attempt to be objective in determining grade values. Our classifiers visited one or two existing small installations but received little assistance because the programming in those installations was being performed by operating officials, which was not in our plan. We evaluated the quality of the duties against the quality of comparable duties in other series such as Methods Examiners, for which standards existed, and determined what we considered to be adequate grade evaluations for our positions. Interestingly enough, the Civil Service Commission subsequently has issued Class Standards for these series and to a great extent incorporated the duties of our positions as typical at the various grade levels to which we assigned them. Our job descriptions have channeled into about every Government agency which either plans to install electronic data processing equipment or has such an operation now in effect.

In the latter part of 1956, as the need for additional qualified employees developed in the electronic data processing operations, invitations were issued again to all employees of the Office of the Treasurer who were interested in being considered for posi-

tions in electronic operations. Seventy-eight employees took the aptitude tests and 16 were selected to attend four weeks of formal training. Upon completion of this training, 7 were reassigned to electronic operations.

In the middle of 1958, a further attempt was made to determine those employees in the Office of the Treasurer who were interested in receiving training in electronic operations. Sixty employees made application, 48 were given an aptitude test, 12 were considered on the basis of scores made in previous tests and a total of 20 was selected for training. Six employees made acceptable grades and were detailed to attend additional courses in programming.

Briefly, in the Treasury, 303 employees were tested for aptitude, 51 were selected for training and 15 successfully completed instructions and were assigned to electronic operations. Of the General Accounting Office personnel involved, 167 employees were tested, 26 selected for training, and 8 finally assigned to electronic operations.

In addition to the aptitude tests, it was mentioned that a supervisory evaluation was obtained on each employee tested; however, the aptitude test was the principal guide for selecting candidates to attend the classes of instruction in programming. Final selections of employees to become regular programmers or console operators were made on the additional basis of marks achieved in Programming School and satisfactory performance of programming duties on subsequent detail assignments.

Generally speaking, the use of the aptitude test as the main guide for selecting employees to receive training in electronic programming has been satisfactory. Our experience establishes the fact, however, that final selection of the employee for regular electronic operations should not be made until the employee has demonstrated acceptable completion of programming classes and progress while detailed to actual programming work. It is interesting to note that, based on our experience, a person who passes the aptitude test with an acceptable rating and has a good background in conventional tabulating operations appears to comprehend a little more quickly the problems inherent in developing computer programs. Another interesting fact is that a number of employees considered by their former supervisors as doing only a satisfactory job are among the best programmers we have developed.

In short, the experience of the Treasury in this matter leads to the conclusion that employees with

an aptitude for programming as indicated by acceptable scores achieved in this aptitude test, and who make acceptable marks in formal classes in programming and who show sufficient interest and effort on their part, have an excellent chance of becoming good electronic personnel.

Probably, the most difficult task we experienced was in translating or defining the requirements of the integrated check payment and reconciliation process to the personnel selected to program the job. It pointed up the necessity for systems people to describe proposed processes in much greater detail than had ever been required previously for conventional type of equipment.

Again for the benefit of those not familiar with Civil Service regulations, we are obliged to work within a framework of rules and regulations which protect the rights of employees to retention under certain conditions. When it is necessary to separate employees because of retrenchment, consolidation of functions, etc., a Federal agency must observe reduction-in-force procedures in determining which employees are to be released from employment. Usually, in private industry when a similar situation occurs, employees may be separated on the basis of seniority primarily, without regard to the type of appointment, veterans preference, etc. In Federal Service employees with a non-permanent type of appointment must be separated before non-veterans with permanent appointments and veterans with permanent appointments. This involves establishing various retention categories and within each category determining relative standing by length of service.

In order to avoid displacing any employees (which would necessitate following the procedure outlined above), we determined to exhaust every other possibility at our command.

First, we reviewed the files of all our employees and categorized them by specializations, i.e., accountants, correspondents, typists, clerks, tabulating equipment operators, etc., based on past training and qualifying experiences. As vacancies occurred within the Treasury first consideration was given to those qualifying for the specific vacancy. This resulted in the reassignment to permanent positions of several employees.

Next, a memorandum was addressed to employees in the Check Payment Division in grades GS-1, 2, and 3, whose positions were being affected by the installation of the electronic system, announcing a refresher course in typewriting for employees having

some basic typing knowledge. The offer for this course, conducted on the employee's own time, by one of our training assistants, brought in about 50 applications. Proficiency tests were given and approximately 30 employees were selected to participate in the course. About 26 participants improved their typing technique to the point that they passed typing examinations and were assigned to positions requiring trained typists or where a knowledge of typing was of value in the performance of the particular duties.

Perhaps in a way this conversion was a blessing in disguise to some of these people. For a number of years they had operated in positions which did not require them to use skills long forgotten and at a level perhaps below their ability. This training was, in effect, a challenge to them to prove what they could do and in some instances strengthened their self-confidence. There were several cases where, because of a lack of urgency, employees continued to perform a rather routine uninteresting clerical function, all the while within themselves harboring a feeling that they were not being used to their full capacity. Being faced with the necessity to qualify for other positions, their hidden talents came to light and resulted in placement in positions for which they are well suited. These employees apparently are contented in their present assignments.

As one reviews the pattern of organizational relationships and the organizational changes required to make possible a system for centralized electronic processing of over 500 million checks annually, it seems remarkable that it was accomplished. It could only have been accomplished by the fullest cooperative spirit of the personnel of many organizational units.

ORGANIZATIONAL EFFECTS

Organization structure is one of the more rigid aspects of administration. This is by reason that changes in such structure are infrequent in most organizations. However, adoption of an electronic data processing system requires basic changes in personnel and procedure which in turn necessitates change in the formal organization structure. Also, substantial changes in working relationships between organizational units must be anticipated. This is the lesson we have learned from the adoption of an EDPS for the payment and reconciliation of Treasury checks.

Principles covering the payment of checks by the Treasurer of the United States:

From the inception of the joint accounting improvement program, instituted by the Secretary of the Treasury, the Comptroller General of the United States, and the Director of the Bureau of the Budget in December 1947, one of the major fields of work has dealt with simplifying and improving procedures and operations relating to government disbursements and collections. A major segment in this area concerns the issuance, payment and reconciliation of more than 500 million checks drawn annually on the Treasurer of the United States by more than 2,000 government disbursing officers. While many improvements were made in the disbursements and collections area during the first few years of the program, it became apparent at an early date that there were real potentials for savings by integrating the check reconciliation operations performed by the General Accounting Office, as a function of external audit, with the payment operations of the Treasurer of the United States. This contemplated a reorganization of the payment function of the Office of the Treasurer of the United States in accordance with the following principles:

- a. It should be a function of accounting and internal control on the part of the Treasury Department, which is charged with disbursement and custody of the public funds, to effect a proof of checks paid in relation to the checks which are issued.
- b. The General Accounting Office, from the standpoint of its responsibilities in connection with accounting systems and independent audit, and the Treasury Department, from the standpoint of its operating responsibilities, should be in complete agreement on the procedures necessary to accomplish such proof of checks paid and the incorporation of these procedures into the accounting system of the Treasury Department as an integral part thereof.
- c. In the light of a revised system of accounting and internal control by the Treasury Department, it should be possible to eliminate the detailed reconciliation of checking accounts of disbursing officers as a function of *independent audit*, substituting therefor

reliance upon the effectiveness of internal control as reviewed in actual operation and the furnishing of such data as may be required for comprehensive audit purposes.

Centralization versus decentralization:

It is significant to record that while the study for installation of the new system was being conducted the predominant emphasis of the joint program to improve the accounting in the Federal Government was one of decentralization of accounting for management. At first glance it might appear that the centralization of check payment reconciliation operations was inconsistent with this general policy and trend. It might be, and has been, argued that reconciliation of disbursing accounts, which involves comparison of paid checks with issue records, is a basic element of internal control of the agency responsible for making the disbursements. Hence, it could be contended that paid checks should be sent back by the Treasurer of the United States (the government banker) to the agencies responsible for making disbursements for reconciliation of their disbursing accounts as a part of the internal control. Fundamental analysis of the problem, however, disclosed that the clerical work involved in handling the processing of paid checks at these diverse points would contribute nothing of substance to the real objectives for decentralizing accounting for management needs. On the contrary, by injecting necessity for the clerical effort involved, it would tend to becloud the real purpose of decentralization of accounting which should emphasize providing management, as a basis for decisions, with useful and reliable data with regard to the programmed and actual costs of the operations for which it is responsible and the effectiveness with which assigned responsibilities are being carried out.

Thus, the centralization of these vast clerical processes involved in the payment and reconciliation of Government checks cannot in any way be regarded as incompatible or inconsistent with the established policy and objective of decentralization of accounting for management. On the contrary, it has facilitated real decentralization in the light of its true purposes.

Impact on organizational structure:

The payment and reconciliation of checks directly involved the Treasury Department, the General Accounting Office and the Federal Reserve Banks.

About 1,575 persons were directly involved in prior operations for payment and reconciliation of checks in these agencies. Of this number about 1,175 were engaged in operations pertaining to the "payment" of checks in the Treasury Department and Federal Reserve Banks, and 400 were involved in processes pertaining to the "reconciliation" of checks in the General Accounting Office. Under the electronic data processing system, the processes of "paying" and "reconciling" checks were brought together in one integrated system in the Treasury Department with a reduction of about 50 percent in overall personnel requirements. This reduction was accomplished notwithstanding that volume has increased in the past decade by more than 60 percent. These data are reflected in Fig. 3.

It is significant to point out here that while there has thus been, in effect, a transfer of processes from the General Accounting Office to the Treasury, there has been no real transfer of functions. The General Accounting Office continues to audit and settle disbursing officers' accounts, based on reconciliations of checks paid against checks issued, and other factors. It has been, however, relieved of the necessity for going through the detailed work involved in reconciling individual paid checks against related check issue records, etc., since this is performed as one part of the integrated electronic payment and reconciliation operation in the Treasury Department. Assurance that adequate controls are built into the Treasury system, as a result of cooperative systems development work and periodic reviews of procedures in operation, provides the basis for eliminating the many detailed processes then performed in the General Accounting Office in connection with its function of auditing and settling disbursing officers' accounts.

It is thus obvious that this change in basic approach to the performance of functions and the related transfer of detailed operations, reduction of personnel and general change in procedure has had a very significant organizational impact on both the General Accounting Office and the Treasury Department. In the General Accounting Office it resulted in the complete elimination of large-scale mechanical operations (on conventional punched card equipment) for reconciling card checks as well as the clerical processing involved in reconciling paper checks. In the Treasurer's Office, where the new integrated operations were established, a complete reorganization was involved. In the Check Payment Division of the Office of the Treasurer of the United

States, the Bookkeeping Branch with 15 employees was eliminated; the Card Check Branch with 49 employees was eliminated; the Electric Accounting Branch was increased from 15 to 50 employees; the Examining Branch with 61 employees was eliminated; The Proving Branch with 69 employees was eliminated; the Reconciliation Branch with 7 employees was eliminated; the Sorting Branch with 44 employees was eliminated; and the Statement Branch with 82 employees was eliminated. However, several new branches were formed: Receiving Branch; Electronic Branch (Data Processing); a new Reconciliation Branch; Files Branch, Control Branch, and a Messenger Branch.

The organizational influence extended far beyond the Treasury Department and the General Accounting Office. For example, provision had to be made for significant and fundamental changes in the processing of government checks by the 12 Federal Reserve Banks and 24 branches. These changes were all in the general direction of simplification. Among other things, the new procedures made it possible to eliminate (1) transfers of various checks from one Federal Reserve Bank to another; (2) the sorting and arranging of checks according to disbursing accounts, serial number, etc.; and (3) the preparation of statements (including listings of paid checks) for various disbursing accounts. These changes stemmed from the fact that under the new procedures all checks are "paid" by the Treasurer of the United States at the central point, whereas under previous procedures most of them were "paid" by designated Federal Reserve Banks acting as agents for the Treasurer. This centralization of "payment" was made possible by use of the electronic data processing procedures for an integrated payment and reconciliation operation and would not have been feasible, because of the large volume involved, with techniques used in the late '40s.

Reorganization of procedures for the issuance of checks:

In order to install the new system, it was also necessary to deal with the problem of integrating the procedures for preparing the checks with the basic changes that had been worked out in the processing of the checks after they had been disbursed. While the procedural changes in this area were not great, they involved the procedures for 2,500 disbursing accounts, which are subject to the administrative control of about 75 Federal agencies. These include

OFFICE OF THE TREASURER, U. S.
 PAYMENT AND RECONCILIATION OF CHECKS
 AND
 PROCESSING CHECK CLAIMS

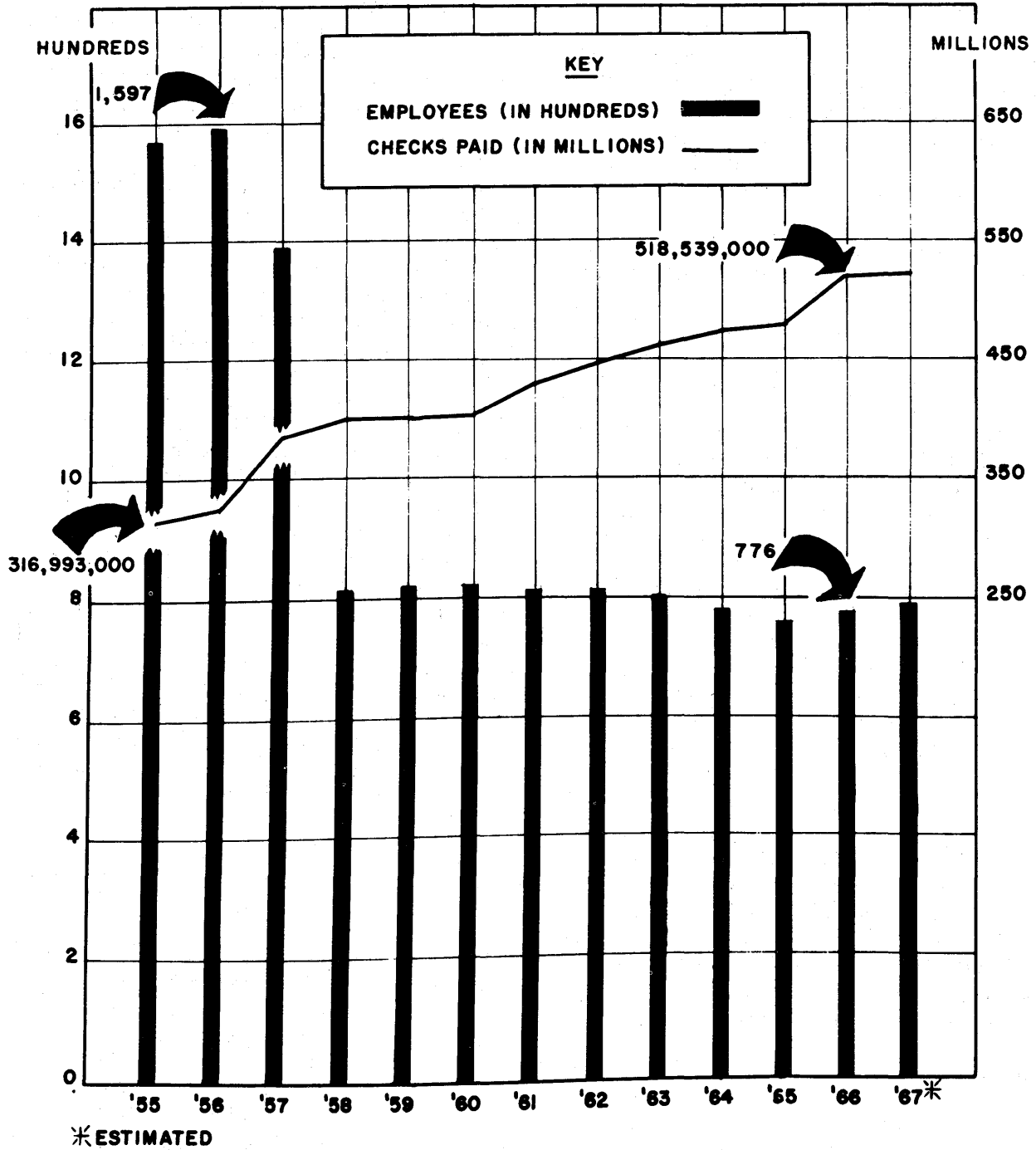


Figure 3. Personnel and volume data.

such far-flung activities as the disbursing accounts of Navy officers aboard ships, and Government officers drawing checks on the Treasurer of the United States in foreign countries.

A key problem in synchronizing check issuing procedures with the revamped procedures for processing checks after they have been disbursed relates to the procedures of those disbursing officers who had not been issuing checks in punched card form. For over four years, representatives of the joint accounting improvement program of the three central fiscal agencies in consultation with representatives of major disbursing agencies where checks were still being issued in paper form—the Department of Defense, the Post Office Department, and certain others—had been working on this problem from two points of view: first, to convert all issuing operations where it was feasible from the standpoint of volume and other considerations to the issuance of checks in fully punched form; secondly, to develop procedures which would permit mechanization in the processing of paid checks for those disbursing officers where it was impracticable to issue the checks in the first instance in fully punched form.

Very substantial progress was made in the first area in bringing about conversion of paper checks to punched card checks. Between 1952 and 1955 an additional volume of about 33 million was converted from paper to fully punched card checks. In 1955 about 12.5 percent of the total number of checks issued was still in paper form. Incidentally, the cost for "paying" this 12.5 percent of the total checks was approximately 63 percent of the total appropriation to the Treasurer for "paying" all checks.

It is, of course, obvious that the electronic data processing procedures for paying and reconciling checks required a solution to the problem of getting the remaining 12.5 percent of paper checks into punched card form so that they would be compatible with the remaining checks. The problem was solved with the close cooperation of the Accounting and Check Subcommittee of the Federal Reserve System. Under the plan which was approved, all disbursing officers for whom it was impractical to install procedures for preparing checks in fully punched form issued a new form of card check which required no punching at the point of issue. From the point of view of the disbursing officer who issues the check, it is inscribed as to payee, amount, etc., as if it were a paper check. These checks are, however, pre-punched at the time of manufacture to identify the

serial number, disbursing office, and other constant information. The amounts are punched by Federal Reserve Banks when they receive the checks through the banking system during the course of their check clearance operations. Thus, when the checks are received at the central facility in the Office of the Treasurer of the United States for electronic processing for payment and reconciliation, they are completely compatible with all other punched card checks.

HARDWARE AND SOFTWARE EXPERIENCES

Early in 1953, the Secretary of the Treasury, the Comptroller General of the United States, and the Director of the Bureau of the Budget, designated representatives to serve as a joint government committee to study the feasibility of utilizing electronic equipment for a large accounting operation of the Federal government.

At the outset, the committee devoted its resources to making a comprehensive study of operations concerning the issuance, payment and reconciliation of the government checks with a view to making recommendations regarding:

1. The use of electronic data processing equipment for an integration of the check payment and reconciliation functions in the operations of Treasurer of the United States;
2. The manufacturer whose equipment is considered best suited for the proposed system; and
3. A course of action, including a timetable and financial factors.

The committee filed its report on September 1, 1955, recommending the use of EDP equipment to perform an integrated function of paying and reconciling government checks within the Treasury Department. The report was approved on October 14, 1955, and the committee was requested to supervise the implementation of the recommended procedures with a view of beginning operations on July 1, 1956. In this connection, budgetary considerations made it desirable to install the recommended system at the beginning of a fiscal year. Perhaps it is appropriate at the outset of a discussion of our experiences in hardware and software to mention the fact that operations commenced on August 1, 1956. The delay of one month was the result of unforeseen problems

involving hardware, training, planning, and many other factors which were encountered in the first eight or nine months of implementing the program.

Request for Proposals:

In the solicitation of proposals, the Fiscal Assistant Secretary of the Treasury wrote a letter to all known manufacturers inviting them to attend a two-day symposium in the fall of 1953. In announcing the symposium, he set forth detailed specifications of present requirements and requested interested manufacturers to submit proposals contemplating the use of machines and components thereof presently being manufactured or under development. Proposals were received from the following:

International Business Machines Corporation
Radio Corporation of America
Raytheon Manufacturing Company
Remington Rand, Incorporated
Underwood Corporation

Criteria:

The committee, with technical advice from representatives of the National Bureau of Standards and the National Security Agency, established the following factors as criteria for evaluating proposals submitted by manufacturers:

1. Reliability and efficiency of equipment
2. Cost of equipment—lease vs. purchase
3. Direct labor requirements
4. Cost of supplies (tape, paper, etc.)
5. Maintenance and service requirements
6. Building specifications and cost of installation
7. Availability of equipment

Evaluation of proposals:

Each proposal was analyzed in detail by the committee, with technical advice from representatives of the National Bureau of Standards and the National Security Agency. Following this detailed analysis, the committee met on numerous occasions with representatives of each manufacturer to discuss in detail certain points of procedure. Early in the evaluation of the manufacturer's proposals, the committee adopted the position that "proprietary interests" would not be permitted from any manufacturer. On

the basis of these discussions, four of the five proposals received were amended by the manufacturers so that they became practically identical insofar as procedural techniques were concerned, although varying as to the specific electronic equipment to be used.

Selection:

The final selection of equipment was narrowed to two manufacturers. The proposal of the International Business Machines Corporation was built around a 705 configuration which resulted in an annual savings of about \$200,000 below the next highest competitor's proposal. Actual tests of live data were performed on both types of equipment and the committee was satisfied that from an operating standpoint either system could adequately do the job. This type of equipment was selected only after detailed evaluation of each of the five proposals received.

Changeover considerations:

The original machine procedures for the payment and reconciliation of checks which were designed around the IBM 705 computer did not differ much from those in use today. However, the demands on the equipment in time outgrew its capacity and processing was transferred to two newly-installed IBM 7070 computers. The two major considerations which led to this changeover to more powerful equipment were: the lack of computer reserve capacity and the increasing difficulties being encountered in servicing check inquiries which had grown to about 1,600 daily in the fall of 1960. Three months of the year, March through May, the IBM 705 computer was in operation three shifts a day, seven days a week, and the annual increase in check volume was about 4 percent. In fiscal 1961 the volume of checks processed was 440 million; in 1962 it ran approximately 458 million; (in 1966 was 520 million). With the IBM 705, the master files had to be split into halves because their volume exceeded machine capacity. Each half was serviced on alternate weekends and thus two weeks might elapse before a report could be furnished on the status of a particular check.

A third consideration was the possibility of savings since the estimates for the new equipment indicated that rental costs would be somewhat less, despite improvements in the quality and quantity of output. Even if additional costs had been involved, however, the first two considerations would probably have led

to a decision to make the change to more powerful equipment, provided, of course, the increased costs were reasonable in relation to the potential improvements.

In planning for the installation of new computers there was one requirement that was paramount: there could be no cessation of operations during the period required for the changeover. Absolute continuity was mandatory. It was obvious, therefore, that a different physical location would have to be found for the new computer system and with the cooperation of the General Services Administration this was arranged.

Programming considerations:

Past experience in programming for the check payment operation prompted a fairly conservative approach with respect to the amount of time that should be provided for testing and debugging programs for the new equipment. Since final testing would involve dealing with large quantities of data, and with large master files which would be converted, it was felt that not less than three weeks of parallel operation should be scheduled for the period immediately preceding the contemplated changeover. In view of these considerations, sufficient funds were budgeted to provide for simultaneous usage of both computer systems for a period of several months. It is fortunate that this approach was taken, since it is now clear that a less conservative one would have led to rather serious difficulties.

At the time the decision was reached to make a changeover to more powerful equipment, the computer system in operation included several thousand reels of magnetic tape, a battery of specialized peripheral equipment and huge master files containing live data. It was most important, therefore, that the new computer system be compatible with the old one, particularly in regard to input and output tape requirements. Initially it was decided to update the system with an IBM 705, Model 3, which would provide urgently needed reserve capacity and at the same time afford an opportunity for improved check-claims servicing. Four months later, after about 40 percent of the programming for the IBM 705, Model 3, had been completed, the new IBM 7070 was announced. The 7070 is a fixed-word-length computer in contrast to the variable-word IBM 705 series computers. To switch to the IBM 7070 required some re-education of the programming staff, systems analysts and a number of the operating personnel. By ordering two of the IBM 7070 computers, not only

could sizable savings be effected but it would be possible to keep workload current by doubling the workload on one computer if the other were down for extensive repair.

Conversion of programming:

The check payment and reconciliation operations in the Office of the Treasurer require six major programs for the IBM 7070s—none of less than one hour duration on the computer—and six major programs for the IBM 1401. Initially programming responsibilities were divided between two lead analysts. One assumed control over the three programs involving Federal Reserve Bank balances, the clearance of checks against the ledger and the stop-payment file, while the other supervised the file maintenance operation—the merging and updating of the main check file and the updating of the outstanding file. The programming effort for the IBM 1401 was also equally divided. Due to the fact that system analysts and programmers were faced with not one but two new computer systems, these responsibilities were realigned as soon as the testing stage was reached to give supervision of the 7070 programs to one and supervision of the 1401 programs to the other.

Program aids provided by the manufacturer proved very helpful in the conversion of the previous IBM 705 programming. One outstanding example is the standard Input-Output-Control-System package program, called IOCS. This standard program contains all of the instructions needed to read and write tape records, including: routines for handling errors; end-of-file and end-of-job routines; tape-labeling routines; and check-point and restart routines. These thoroughly-tested instructions comprise about 40 percent of those required in most large programs and may represent as much as 60 to 80 percent of some. Their use not only enabled the programming staff to devote more time to the other areas but provided a standard input and output routine for all programming. Standard routines also simplified the console operations.

In the programming conversion, the search for the most efficient programs was extremely thorough. When two million checks are processed daily, and this was a normal volume during the income-tax refund period, there is a daily total of 58 million operations. The records are read from tape into the computer ten times, processed ten times, and written from the computer onto output tape nine times—a

total of 58 million operations. If only 144 micro-seconds per operation were saved in each program, about \$4,800 would be saved yearly. Since each of the six major programs has from 4,000 to 8,400 instructions, the potential for savings was quite substantial. We used symbolic language throughout all of our programs.

Conversion lessons:

On the whole, the conversion undertaken in the Office of the Treasurer of the United States to change electronic processing from IBM 705s to IBM 7070s and an IBM 1401 was extremely interesting and highly satisfying. From the experience gained in this undertaking it can be said that anyone concerned with the conversion of a computer system would do well to consider:

1. Teaming an inexperienced or newly-trained programmer with an experienced programmer.
2. Giving careful consideration to master-file conversion programs.
3. Using good personnel on all assignments.
4. Guarding against overeagerness to test the new computer.
5. Allowing the engineers ample time to check the equipment thoroughly before taking over.
6. Spending as much time as possible in checking programs before actual machine testing is begun.
7. Planning parallel test runs to include as much volume as possible.
8. Providing generous allotments of time in planning conversion schedules—unforeseen problems do arise and machine failures on new equipment must be expected.
9. Using packaged programs furnished by the computer manufacturer and assigning a programmer to familiarize himself with each—they should become experts in order to see that the systems function properly and are currently maintained.
10. Avoiding undue overtime for any additional programmer—a tired programmer can cause much damage.
11. Having the machine testing controlled by one person who is made responsible for scheduling on a priority basis and for coordinating all operations.
12. Verifying the results of the parallel operation in minute detail—overlooking a seemingly minor detail may be costly.

PROCESSING POSTAL MONEY ORDERS

The Treasurer's electronic data processing facilities are also being used to service the Post Office Department's money order operations, on a reimbursable basis. This program involves the use of procedures and techniques which are quite similar to those described for the check payment and reconciliation operation. A major variation, however, is that the accounts of some 35,000 postmasters are involved, who issue approximately 210 million money orders annually. Our review indicates that about \$750 thousand is being saved on a recurring annual basis through implementation of this plan.

A LOOK INTO THE FUTURE

I have devoted considerable time to discussing the past and present systems for payment and reconciliation of government checks. Perhaps a peek into the future would be helpful. One might ask "How can you improve the present system?" Certainly we can't eliminate the one remaining handling. If we could, we wouldn't need checks at all. I'm sure you all have heard or read about the no check—no cash economy—or—the universal credit system. Much already has been written and much more will be. The day may come when we can eliminate or practically eliminate checks.

Undoubtedly, the technology to develop a universal credit system is available today. Whether such a system could be justified from an economic standpoint today or ever is another question. In order to explore the question of economic justification, one must first determine who among the users stands to benefit economically from a universal credit system. The users are merchants, banks, and customers. The benefit to customers is debatable when compared to today's credit system. Sure, it would eliminate the necessity for him to write checks, but at the cost of immediate loss of cash in his "no check" account at his bank. Of course, if the system provided for automatic overdraft coverage by the bank at an agreed rate of interest, the customer would pay a carrying charge (to the bank) for many purchases which are interest free to him inasmuch as he now pays by

check some time after he has deposited funds to cover such payments.

It will be contended that the immediate availability of credit will bring the merchants' prices down and the customer will benefit from reduced prices. This probably would be true if the system were truly universal and used by everyone. In other words, all merchants would be on a cash basis if immediate charge were made to the customer's bank account. I would, however, remind you we haven't yet considered the cost for this super network of communications linking millions of input-output gadgets. It looks as though customers would not benefit economically, at least the initial studies do not demonstrate that it would have any immediate economic value to customers.

The other users of such a system are merchants and banks. Both of these users have a motivation (profit) to the establishment of such a system. However, in the final analysis the customer's desire in the matter will prevail. I suggest that he will, at the outset, have the option to be billed monthly as at present or automatically to his bank. Bank billing may be daily or periodically. If these basic assumptions are correct, it would appear that a universal credit system will emerge only when the cost of operations is not prohibitive. Assuming that a system could be devised which would be profitable for the owners, the question then is "Who would be the owners?" Either banks or merchants or perhaps both. Will there be competitive systems similar to those in existence today or will there be a single one? Many questions are still unanswered and much more study must be given before we see the real beginning of a universal credit system.

Perhaps there may be something more important than the profit angle. I have in mind that any such study should possibly consider whether the government should operate such a system with eventual ownership passing to the public. I am not advocating government ownership. I am merely suggesting that consideration be given to such an arrangement. The government certainly has a major interest in credit and monetary policies of the country. I would suggest, in closing, that the establishment of a universal credit system will pose a lot of questions and problems with which the government has a concern.

Regardless of who owns and operates the system the initial step is to agree on the method of identification. The social security number, with the addition of a self-checking digit assigned at the time of birth, seems to me a basic requirement if we are ever going

to have a no check—no cash economy. If this step is not forthcoming, then I fear we will continue to romance with the idea. Both within and outside the government, this idea will continue to require study, research and development.

APPENDIX A

GENERAL OUTLINE OF PREVIOUS PROCEDURES

The following outline is a generalization of the principles observed in the Federal Government prior to adoption of the electronic system for the payment and reconciliation of Government checks.

The Treasurer of the U. S.:

In the Federal Government, the Treasurer of the United States occupies the same relative position in relation to an authorized government disbursing officer as the bank does, to the holder of a commercial or personal checking account in the business world. Checking accounts are established on the books of the Treasurer of the United States for those individuals who are authorized by their respective government agencies to make disbursements of government funds. Checks drawn by these individuals are "paid" only after they have been examined by the Treasurer's Office and charged against the appropriate checking account on substantially the same basis as checks paid by a commercial bank.

Issuance of checks:

Checks drawn against the Treasurer of the United States can only be issued by authorized "disbursing officers." For the greater majority of civilian agencies the issuance of checks is performed by another organizational unit of the Treasury Department—the Division of Disbursement. This Division maintains regional offices throughout the country where checks are issued on the basis of certified vouchers submitted by the agencies which incurred the obligations for which the payments are made. In the military departments and certain civilian agencies the disbursing officers are attached to the operating agencies which incur the liabilities which give rise to the payments. Such disbursing officers are located throughout the world but for each a checking account is established on the books of the Treasurer of the United States and all checks drawn by such officers are ultimately

charged against such accounts in the process of "payment." In all, about 2,400 checking accounts are maintained on the books of the Treasurer of the United States.

Each authorized disbursing officer is provided with an appropriate stock of blank checks, each of which carries the designation "TREASURER OF THE U.S." (at the place where the name of the bank usually appears on a commercial check). The checks also indicate in each case the disbursing symbol (identifying number) of the checking account on the books of the Treasurer of the United States against which the checks will be drawn. Most government checks were issued in punched card form (about 300 million out of a total of 350 million a year). Such checks were normally pre-punched at the time of manufacture with the disbursing officer's checking account number and the identifying serial number of the check. During the process of issue the amount and date of issue (and in some cases certain reference information relating to the disbursement) were punched into the check in addition to being inscribed on the face. Prior to installation of the new system, where it was impracticable or uneconomical (by reason of low volume or otherwise) to install punched card equipment at the check issue points, conventional paper checks were issued.

Basis of control over check-issuing operations:

The following is a brief summary of the principal features of the general plan of control over check-issuing operations which are important from the standpoint of a general understanding of the controls surrounding the check issuance, payment and reconciliation processes in the Federal Government:

a. The disbursing officer is held accountable for all blank check stock with which he is supplied. That is, he is required to control the use of his stock, and make periodic accountability reports (which are subject to both internal and external audit), so that he can account for all check stock received as either (1) issued, (2) canceled or spoiled, or (3) on hand.

b. The disbursing officer is required to support all checks issued by vouchers approved by an authorized "certifying officer" of the agency for whom he makes the disbursement, and his accountability for issuance of checks is determined on that basis. In this connection he prepares a monthly report (commonly referred to as his "Account Current") which shows, among other things, the total (supported by a listing) of (1) the checks issued (usually a copy of a machine

run showing check number and amount) and (2) the total of the certified vouchers he has as authority for such disbursements supported by the originals of the "certified" vouchers on the basis of which he made the payments. The procedures established for (1) controlling the "certifying officer" (i.e., with respect to the underlying legality, propriety, etc., of the authorization for the disbursement represented by the voucher) and (2) the comparison of the disbursing officer's record of checks issued with the related authorizing vouchers are beyond the scope of this discussion. We are concerned here with the procedures involved in the final step in the control of the disbursement process—i.e., proving through appropriate "reconciliation" procedures that the checks as actually "paid" are in agreement and reconciliation (through development of outstanding checks, etc.) with the checks reported by the accountable disbursing officers as having been issued and pinning down responsibility for any discrepancies.

General flow of government checks through commercial channels:

Checks when issued are normally mailed directly by disbursing officers to the payees indicated. From the payees they, of course, find their way through normal business channels to a commercial bank. The commercial bank in turn sends the checks it receives to an authorized government depository (normally a Federal Reserve Bank). The Treasurer of the United States maintains funds on deposit (in an account known as the "Treasurer's General Account") at each of the thirty-six Federal Reserve Banks against which the checks can be charged as a basis for enabling the Federal Reserve Bank to extend immediate credit to the remitting commercial bank. The Federal Reserve Bank then sends the checks for "payment" to the Treasurer of the United States.

Former decentralization of "payment" function:

Due to the large volume of work involved (well over an average of a million checks a day), the Treasurer of the United States found it necessary under procedures previously followed to decentralize the "payment" function so that this large work load could be distributed to a number of different places. This was accomplished by designating various Federal Reserve Banks as her "paying agent" for specific checking accounts. Under this arrangement each Federal Reserve Bank maintained the accounts and per-

formed the related "paying" functions for a designated group of checking accounts. All checks drawn on such accounts carried, in addition to the Treasurer of the U. S. designation, the legend "Payable Through Federal Reserve Bank of _____." Under this general arrangement, each Federal Reserve Bank was required each day to sort the Treasury checks it received from commercial banks according to the checking accounts against which they were drawn as a basis for enabling it to send them to the proper point of "payment" (sorting of punched card checks was, of course, done by machine; paper checks by hand). Those drawn on checking accounts for which the receiving Federal Reserve Bank was itself the paying agent were, of course, retained by the Federal Reserve Bank to which they were sent by the commercial bank. Checks drawn on other checking accounts for which other Federal Reserve Banks, or the Treasurer of the United States in Washington, were the designated points of payment were forwarded to such points. All checks received by each Federal Reserve Bank (or the Treasurer of the United States)—whether direct from commercial banks or from other Federal Reserve Banks—were combined for processing through the "payment" procedures outlined below.

Former "Payment" Procedure:

The following are the principal elements of the "payment" procedures as previously followed by the Federal Reserve Banks or the Treasurer of the United States for checks drawn on the checking accounts for which they were the responsible paying agents:

- a. All checks were sorted by serial number within the checking account symbol against which they were drawn (by machine in case of punched card checks; manually in case of paper checks).
- b. Active "stop-payment" notices were checked against the checks presented for payment by serial number and any checks thus intercepted returned through banking channels to the remitting bank.
- c. Checks were examined for genuineness of drawer's signature, evidence of alteration, etc.
- d. Checks in order for payment were listed (by tabulating machine in case of

punched card checks and adding machine in case of paper checks) and related totals developed for posting to the checking account. This list served as support for the statement of the checking account (comparable to the customary bank statement of a commercial bank). It showed both the identifying serial number and amount of each check and was used for reference purposes for handling inquiries regarding claims of non-receipt of checks by payees and related requests for stop-payment, issuance of duplicate checks, etc.

Former Procedure for Reconciliation of Checking Accounts:

As in commercial practice, paid checks and related statements of checking accounts were sent from the point of payment (the bank in private business; the Treasurer of the United States or the paying Federal Reserve Bank in the Federal Government) to the point where a reconciliation of the statement could be effected with the corresponding records of checks issued. In business this is generally done in the accounting department of the business whose checking account is involved. Such reconciliation is, of course, subsequently reviewed as a part of whatever independent audit is conducted of the firm's books as one phase of the review of internal checks and controls. In the Federal Government this reconciliation was heretofore performed centrally directly by the auditing agency—the General Accounting Office—as a part of its responsibilities for auditing and settling disbursing officers' accounts for their accountability for proper disbursement of government funds. The extensive use of punched card checks made it possible to place a substantial portion of these operations on a highly mechanized, mass-production basis with the use of conventional punched card equipment. The reconciliation was ordinarily performed about three months after the close of the month of issue (at which time there were normally very few outstanding checks). The monthly statement of disbursing officers' accountability (referred to in paragraph 3b) and which is supported by lists (i.e., usually copies of machine runs) of checks issued formed the basis for reconciling the statements of the paying agency (i.e., the Treasurer of the United States or her agent Federal Reserve Bank) with the

accountability records of the disbursing officer who issued the checks. Statements of differences, etc., developed during the course of these reconciliations, lists of outstanding checks (for reference use in processing claims for duplicates, requests for stop-payment, etc.) were supplied the issuing and paying agencies and appropriate adjustments effected.

Relationship of checking account reconciliation to other control and audit processes:

This paper deals only with the check payment and reconciliation processes. It is obvious, however, that these processes are but a small part of the total and much broader problem of control and audit of financial operations. In recent years the emphasis in the Federal Government has been on providing effective internal controls over all financial operations in the accounting systems and related procedures of the individual responsible operating agencies. External audit by the General Accounting Office is to a constantly increasing extent performed on the basis of a review of such internal controls and selective examination of individual transactions—employing much the same basic approach used by public accounting firms in the audit of commercial enterprises. The central reconciliation of checking accounts is, of course, readily coordinated with this broader audit (through the monthly accountability report of the disbursing officer) and avoids the necessity for much detailed clerical work at the sites of operation.

APPENDIX B

SYNOPSIS AND COST ANALYSIS OF EDP PROGRAM IN THE OFFICE OF THE TREASURER, UNITED STATES

In June of 1953, a committee was established, composed of representatives of the Bureau of the Budget, General Accounting Office, and Treasury Department, to study the feasibility of using electronic equipment for handling Government checks. Sixteen manufacturers of electronic equipment with the required capacity potential were requested to submit proposals, including the cost of equipment recommended. Proposals were received from five manufacturers and selection of equipment was made on the basis of comparative costs.

Conversion of check payment and reconciliation operations to the electronic system was started in August 1956 and completed in January 1958. Be-

fore conversion to the new system the committee estimated there would be an annual recurring savings of \$2.2 million. A comparison of costs for fiscal year 1959, the first complete year of operation under the new system, with costs under the old system in 1956, showed an annual recurring savings of \$2.9 million.

In March 1960, the Post Office and Treasury Departments initiated a study to determine the feasibility of expanding the electronic facilities in the Treasurer's Office to provide for processing postal money orders. Conversion of the money order operation to the electronic system was started in June 1962 and completed in April 1963. This resulted in additional savings to the Government of \$750,000 annually.

With the approval of the House and Senate Appropriations Subcommittees, a capital investment of \$2½ million was made during fiscal years 1963 and 1964 to purchase the electronic equipment. After recovery of this capital investment, which will occur this year, an additional annual savings of \$900,000 will be realized as a result of purchasing the equipment compared to what it would have cost to rent the equipment.

At the present time, the existing electronic system is considered adequate to meet the needs of this office in the foreseeable future. However, our representatives attend meetings at which are demonstrated advances in electronic machines and techniques of different manufacturers. Thus far, nothing has been developed which would improve our existing system in terms of service or cost, but we will continue these appraisals.

SHARING EDP EQUIPMENT

Amortization costs recovered from other bureaus and agencies for the use of our purchased equipment from July 1, 1962, through December 31, 1965—amounts recovered and deposited to the general fund as miscellaneous receipts are shown in Table I.

ACKNOWLEDGMENT

The success of this application, embodying numerous technologies, is the result of the dedicated effort of many individuals. To these individuals the author extends his sincere appreciation. I wish to also acknowledge the contribution of Professor Walter Frese of the Harvard Graduate School of Business Administration who as Head of the Accounting Systems Division of the General Accounting Office was large-

Table I

Bureau or Agency	Fiscal Year 1963	Fiscal Year 1964	Fiscal Year 1965	Fiscal Year 1966 through Dec. '65
Post Office Department	\$37,000	\$151,219	\$168,416	\$80,400
Agriculture Department	17,841	53,615	61,286	14,848
Railroad Retirement Board	3,200	5,801	6,604	3,200
Federal Reserve Board	—	83	2,575	—
Labor Department ..	5,433	—	247	—
Veterans Administra- tion	50	5	—	—
Navy Department ...	17	—	—	—
Treasury Department:				
Internal Revenue Service	2,358	1,135	1,688	—
Bureau of Public Debt	—	505	700	—
Bureau of Accounts	—	440	634	—
Office of the Secretary	—	343	328	—
Office of Interna- national Affairs ..	—	170	203	—
Comptroller of Currency	—	—	5	—
Total	\$65,899	\$213,316	\$242,686	\$98,448

ly responsible for organizing the original study. His counsel to the committee was a major contribution to the success of the study.

BIBLIOGRAPHY

1. "Electronic Processing of U. S. Treasury Checks," Interim Progress Report, Washington,

D.C.: Joint Government Committee Representing the Bureau of the Budget, the General Accounting Office, and the U. S. Treasury Department under the Joint Accounting Improvement Program, June 30, 1954; 119 pages.

2. "Electronic Processing, U. S. Treasury Checks," Final Report (Equipment Acquisition Proposal), Washington, D. C.: Joint Government Committee Representing the Bureau of the Budget, the General Accounting Office, and the U. S. Treasury Department under the Joint Accounting Improvement Program, September 1, 1955; 84 pages.

3. "Final Joint Report on the Electronic Data Processing Installation in the Office of the Treasurer of the United States," Washington, D. C.: Joint Government Committee Representing the Bureau of the Budget, the General Accounting Office, and the U. S. Treasury Department under the Joint Accounting Improvement Program, August 28, 1959; 8 pages.

4. Hearings Before the Subcommittee on Census and Government Statistics of the Committee on Post Office and Civil Service, House of Representatives, Eighty-Sixth Congress, First Session, June 5, 1959.

5. Hearings Before the Subcommittee on Census and Government Statistics of the Committee on Post Office and Civil Service, House of Representatives, Eighty-Sixth Congress, Second Session, March 2 and 4, 1960.

6. Hearings Before the Subcommittee on Census and Government Statistics of the Committee on Post Office and Civil Service, House of Representatives, Eighty-Seventh Congress, October 2, 3, and 5, 1962.

7. "Use of Electronic Data Processing Equipment in the Federal Government," House Report No. 858. Committee on Post Office and Civil Service, House of Representatives, October 16, 1963.

8. Robert H. Gregory and Richard L. Van Horn, *Automatic Data Processing Systems: Principles and Procedures*, 2nd Edition, Wadsworth Publishing Company, Inc., Belmont, California, 1963, 816 pages.

PROBLEMS OF INFORMATION SYSTEMS IN STATE GOVERNMENTS

Dennis G. Price

*Director, State Computer Systems Development
Albany, New York*

DEVELOPMENT OF STATE GOVERNMENT INFORMATION STRUCTURE

The Cause of the Information Structure

Like all forms of government organization, the structure of state government organization has evolved over the years, responding to the new demands of the people served. Because the structure has had to adapt to these demands, generally there has been a piecemeal development, and it has been virtually impossible at most times for the structure to evolve according to any long-term plan. Of course, periodically there have been comprehensive investigations into the structure at any given time, but these in general have not resulted in massive reorganizations.

The easiest way to respond to a new demand for services is to create a special organization whose function is solely to administer to that demand. This has the advantage of giving the new function emphasis, making it clearly visible, and pinpointing the responsibility. Very often, therefore, new agencies have been created in response to demands for these services. If one were to look at the organization chart of practically any state government one would find departments such as Mental Health, Welfare, Motor Vehicles, Commerce, etc.—functions which

usually did not exist 50 years ago. And these are clearly labeled as separate departments. In addition, there is usually a proliferation of committees, commissions, authorities, and other groups of a temporary or permanent nature which reflect new and pressing demands for services.

The Response of State Governments

The response of the newly created organizations in terms of information has been simply to request new information! It is a cliché that information is the lifeblood of an organization, and there is no doubt that the creation of a new agency, usually in response to some crisis demand, leads to a request for new information quickly. This traditionally has had the following effects:

1. *Duplication of Information.* Although the new agency may be aware that the information it needs is available in another government agency, it does not get it from this other agency for two basic reasons. One is that the information is not in the actual format required. And the other is that the information may not even be immediately accessible without cutting through masses of bureaucratic red tape. For these reasons, the new agency prefers to get its own information.

2. *Increasing Requirements at the Source of Information.* Because the new agency makes pressing demands for information in order to provide the services, it goes immediately to the source of the information and requests this source to cooperate. Even though the source is providing the same or virtually the same information to another agency, it has very little choice other than to provide the information as requested. This, of course, is very unsatisfactory and creates resentment at the source of information—the taxpayer, other governmental organizations, or private industry.

3. *Increasing Difficulty in Relating Information Concerning the Entity.* Because the source of information has given the same information to two or more agencies, it now becomes increasingly difficult for these agencies to relate information concerning the same entity. Had the information been sent only once, then of course this difficulty could have been avoided. Unfortunately this is not the way matters develop. Therefore, we arrive at a situation where essentially the same information is available in several agencies of government, derived from the same source, possibly in different reporting periods, and probably in different formats, so that it is almost unrecognizable as the same information when it is received and processed by the receiving agencies.

4. *Increasing Cost of Information.* Since the information is generated several times from the same source, and is processed by several agencies, obviously the cost of that information is much higher than it need be. Moreover, because the information is now stored in incompatible formats, another cost is added: the cost of reconciliation.

Recent Examples of New Demands for Service

We are all aware of the problems of our growing urbanization, the dynamic growth of our economy, and the realization of the goals of equal opportunity. When state governments act to solve or alleviate these problems we see the growth of new functional agencies. These are concerned with law enforcement, economic opportunity, unemployment, education, health, urban redevelopment, and statewide economic planning. All of these in turn create problems of how and where to find new revenues.

They also create problems of how and where to find the required information.

FEDERAL-STATE-LOCAL GOVERNMENT INFORMATION RELATIONSHIPS

The demands of the federal government for information constitute a growing problem for the states. It is interesting to note the reasons for these demands. Historically the federal government has been primarily concerned with fiscal accountability, as one of the tunes called by the federal piper in return for federal funds. Secondly, and more importantly today, information from the states reports to the federal government the progress of application areas and programs for which it has responsibility or for which it shares responsibility with the state government. Thirdly there is a mutual interest shared by the federal government and by the state government in having the federal government collect information from all states and then disseminate it.

If we were to examine the various federal programs, we would note that they had grown up in piecemeal fashion, so that they have established their own reporting requirements with the corresponding state and local government agencies. This in turn means that much information is sent by the state to the federal government without its being related to the needs of other state agencies.

Some examples of federal programs which are having a major impact on state information collection follow:

1. *Education.* The Office of Education has developed a "basic educational data system" (BEDS) which is aimed at collecting source data from the records of educational agencies and institutions. This information will flow to the state departments of education and from there to the Office of Education in machine-readable form. The BEDS system will contain basic data in all five areas of staff, instructional programs, pupils, finance, and facilities. It will furthermore be compatible, collected at the same time, and therefore most useful to the federal and state authorities.

2. *Public Health.* A very considerable amount of information is reported on vital statistics, these records being created in about 30,000 local offices and passing usually through the state health departments. Practically the whole burden of maintaining the vital statistics system is borne by the states—an example of cooperation rather than compulsion in federal reporting.

3. *Criminal Information.* Since 1930, when the Uniform Crime Reporting Program was instituted, a

considerable amount of information on criminals and criminal activity has been collected by the FBI. Again this very successful program has been maintained on a voluntary basis. Beginning in 1967, the National Criminal Information Center in Washington will receive information from cooperating state and local governments concerning stolen vehicles, certain stolen property, and extraditable warrants for wanted persons. The information will be sent in machine-readable form via communication facilities. Any cooperating government can then inquire of the file.

The above are only a few of the many applications areas with which federal and state governments are concerned. Obviously such new major federal programs as pollution control and medicare will have and are having equally significant effects. The major deterrent to the flow of information from the local to the state and from the state to the federal level is simply the incompatibility of the data. In other words, the same data in many application areas are coded differently by local governments, states (and even by different agencies within the same state), and by the federal government.

Thus, there is a need for the standardization of data elements and their codes before an efficient information flow can take place among the different levels of government. This problem is being now widely discussed, and in a few instances some government jurisdictions are taking action. For example, the Tri-State Transportation Commission (which is financed by the three states of New York, New Jersey and Connecticut, the Bureau of Public Roads and the Department of HUD) is responsible for transportation planning in the New York City conurbation. It spent a great amount of time developing a land use code, a geographic location code, and other codes, simply because there were no standard codes available.

There are some helpful signs for tackling this most significant problem:

1. The February 1966 National Conference on Comparative Statistics, which was held in Washington and attended by government officials from all levels of government, discussed this major problem.
2. The Council of State Governments' Committee on Automation, Technology and Data Processing has been well

aware of the problem and has given its support to the American Standards Association's efforts in the standardization of data elements and their codes. It has also given its endorsement to the establishment of a joint data processing center in Des Moines, Iowa, which would process local, state and federal data; part of this effort would, of course, be concerned with data standardization.

3. GSA is currently considering a proposal to establish a joint local-state-federal data processing center.
4. There is a bill (S. 561) that has passed the Senate and gives all federal agencies, rather than only a few, at present, authority to cooperate with state and local governments on technical matters, including data processing.
5. The Conference of Governors has recommended that central statistical offices be established in each state. New York State has had such an office for two years, and it has made a complete inventory of statistical series maintained by the various state agencies.

Of course, while the federal government must take the lead in the national programs, the state governments should similarly take the lead in educating and assisting the local governments. The present state-local situation is similar to that existing at the federal level where we have noted a piecemeal approach. Presently, most state governments, which are doing anything about assisting their local governments, are using a rather fragmented approach which concentrates on the mechanization of specific functions. This makes it difficult for a municipality to consider the use of one computer for all functions, as against several computers or the use of service bureaus, and tends to inhibit the design of "total" systems. Moreover, the regional concept of providing cooperative data processing services cannot be given proper consideration if a piecemeal approach is followed.

Presently in New York State there are five or six state agencies which are all concerned with giving consulting advice on data processing to local governments. Recently we have coordinated these ap-

proaches by establishing a Joint State-Local Automation Planning Council, composed of senior officials from state and local governments. It is developing a comprehensive program in functional areas (such as welfare and education) and planning for the establishment of regional computer centers and the provision of expert advice in technical data processing areas.

California has taken a similar approach using the concept of an "information central" connecting all state and local government computer installations. Of course, for the local jurisdictions to be able to talk to any state agency computer via the information central, standardization on data elements is essential.

CENTRAL CONTROL OF INFORMATION SYSTEMS DEVELOPMENT IN STATE GOVERNMENTS

Because of the foregoing discussion concerning the development of state government information structure, the impact of the federal demands, and the work of the states with their local governments, it is obvious that there is a very large investment currently in state data processing. About two years ago I estimated that all the states together were spending \$25 million on computer rental, and, when the costs of associated peripheral equipment and personnel were added, the figure was between \$80 and \$100 million annually for data processing activities. The figure today is probably between \$120 and \$160 million.

In order to control these investments many states have adopted very positive programs by establishing

a central agency. In New York State for example the Division of the Budget has developed this control and guidance along several lines to assist line departments in all aspects of computer acquisition and use—for example, in the areas of information studies, issuance of specifications, realistic evaluation procedures, monitoring installations, development of a central computer, etc. Similar developments have taken place in such states as Texas, Hawaii, Michigan, Wisconsin, and Ohio, and we have already mentioned the work of California, which is now proceeding apace.

In our own state, we have recently developed a statewide master plan for data processing development, which includes a data element inventory for all-important master files, either mechanized or not, throughout the state. This inventory will be circulated to all potential users, and we shall soon be implementing a much more effective interagency use of data with the accompanying standardization of data which this entails. The plan also encompasses the development of new computer applications for the next five years for state departments, and as such plots out their development in the data processing area.

It is hoped that with such developments as these by leading states in the country, and with the leadership provided by the federal authorities, we shall within the next decade approach a position where the important data which flows among governments will have been standardized, and we will at last be in a position to "talk" to one another by computers—as has been the dream of practitioners for the last decade.

THE IMPACT OF COMPUTERS ON LOCAL AND REGIONAL GOVERNMENT

Herbert H. Isaacs

*Isaacs Research and Consulting, Inc.
Los Angeles, California*

INTRODUCTION

Whenever we talk of the "impact of computers" on a given area of applications, we usually become concerned with the technical character of the applications and how they might be supported with computer systems and/or other advanced information processing devices. Thus, in viewing the computer's impact on local and regional government, we could easily focus our discussion here on techniques of water billing, crime information retrieval or regional economic data analysis. And, indeed, some attention must be given to matters of this sort.

We have observed, however, that whatever the potential impact of the computer on local and regional government might be, that impact has been somewhat retarded up to this point. In looking for reasons for this retardation, two major problems are apparent. First, the information specialist is usually not familiar with the nontechnical influences on the successful introduction of new technology. And second, both specialist and administrator tend to focus on limited individual applications instead of dealing with broad functional areas where more sophisticated techniques and devices might be applicable.

This paper is an attempt to explore the present and future impact of the computer on local and regional government. Such an objective certainly requires a look at the history of computer uses in local

governments and a projection of new technology's future applications. But considering the two problems described above, it would appear desirable to first explore the organizational and functional contexts in which computer applications must fit.

Accordingly, the paper is organized as follows. The first section provides some background information on the organization of government in local areas, and the intergovernmental relations among state, federal and local agencies and political jurisdictions. These factors have a great influence on what projects are funded, and often dictate the technical content of those projects.

The second section describes the major functional areas of local and regional government. It is within and across these areas that particular computer-oriented projects apply.

In the next section, we describe the past introduction of computers into local government, and give some examples of more current projects of a demonstration or developmental character.

In the final section we provide some projections for the future. We describe there some of the potential impact of new computer hardware and software on the technical, administrative and political factors that have heretofore impeded rapid progress. We conclude with some considerations concerning needed software developments.

ORGANIZATION AND INTER-GOVERNMENTAL RELATIONS

The functional responsibilities of local government are essentially delegated by the state government. The U.S. Constitution provides specifically for the states to undertake those functions necessary for the general safety and welfare of its citizens. Each state deals slightly differently with this problem but, in most states, local governments are franchised by the state, either through charters or other state constitutional provisions.

These considerations of responsibility are important in dictating how overall programs are initiated, decisions made, and funding administered. For example, the California Statewide Federated Information System proposed in the Lockheed study¹ was specifically designed to support the concept of "home rule," implicit in California's local government and state relations. Other states may exhibit more state-centralized control of functional areas, but in general, the basic operating functions of a city (such as public works, police, fire protection, health, sanitation) are provided by some form of local government.

These on-going services and functions are usually supported by local taxes, particularly the tax on real property. Some local and regional programs cross local jurisdictional boundaries; for example, education, highway projects, and certain welfare programs. Funding for these kinds of programs is provided from a combination of local and statewide sources, with state funds being allocated back to the local areas according to some agreed-upon formula.

Federal programs concerned with problems of the metropolitan area can be channeled through both local government and statewide agencies, depending upon the enabling legislation. Most Federal programs, however, involve some tacit consent of the state executive, even for programs administered directly at the local level.

Before we leave the subject of organization, it is important to note that the concept of *regional government* is in most cases just that—a concept. Strong support for regional government in metropolitan areas is reflected in public administration and political science literature.² Nevertheless, in actual fact, existing centers of local political power tend to resist any form of more centralized control which would reduce the power of a particular local jurisdiction. As a result, regional planning organizations

tend to exist on a cooperative basis, with authority only to set standards. They do not usually possess the power to enforce those standards.

The issue of whether or not regional government would be more effective in dealing with metropolitan problems is quite controversial. Differing technical as well as political views on this subject may be found. One point is clear, however. The introduction of technology, and particularly information technology, into metropolitan government is greatly inhibited by the multiplicity of decision points involved in getting a program approved and operating successfully.

FUNCTIONAL APPLICATION AREAS

The following is a brief summary of the different application areas in local government. A thorough treatment of all the activities of each agency and department serving functions in metropolitan government would and, in fact, does take volumes (of organizational manuals and procedures). Nevertheless, even this surface description will give some idea of the great diversity of activities being carried on by local government.

For purposes of presentation, the activities are grouped in seven general categories: agencies that are facilities-oriented; those that are people-oriented; agencies concerned with safety; with planning; with administration, tax and fiscal matters; individuals and groups concerned with policy-making; and, certain other nonlocal government agencies that have functions bearing on the local government problem (and may even occupy offices in local areas).

The scope of this paper does not permit a detailed description of the information processing support required for each operational function. However, certain areas have been emphasized as examples of typical requirements.

Facilities-Oriented

In this group we include: (1) the public works departments concerned with major plant facilities, buildings, streets, water and sewage networks, as well as supporting services such as engineering to maintain and expand those facilities as necessary; (2) the building and safety departments concerned with the approval and inspection of private facilities and property; (3) the utilities, such as water and power, either publicly or privately owned; (4)

recreation and parks departments; (5) public transportation agencies; and (6) traffic departments concerned with the flow of vehicular traffic, its control and safety.

People-Oriented

The people-oriented agencies include: (1) the public schools; and (2) various community service agencies and welfare departments. Activities of the schools are not confined merely to the employment of teachers and setting of curricula. The administration of a school system includes such complex activities as areawide facilities planning and specific building construction. Similarly, the community service and welfare agencies are engaged in large scale field activities, as well as interviewing and screening in field and central offices.

Safety

The traditional agencies concerned with safety include: (1) police; (2) fire; and (3) health departments. The police are engaged in a multitude of patrol and investigative functions under an ever-increasing crime and traffic load in the metropolitan areas. To support this patrol and investigation, large volumes of records are kept concerning specific events and people connected with those events. A major communications and dispatching function is also required. Supporting the overall operations are the logistics and management information needed to monitor and allocate scarce resources.

The fire department has a similar on-line dispatching requirement. Current information is also maintained on the building characteristics of large businesses and industrial plants so that rapid, effective action can be taken in the event of a fire. Furthermore, the incidence of fires across the city must be evaluated in order to allocate equipment and personnel resources and to plan for the construction of new facilities.

The health department has a similar dual need for both operations and planning. It is particularly important that the health department be able to isolate disease trends before they reach epidemic proportions.

All the safety agencies, including the civil defense staff and responsible city executives, encounter an especially complicated problem in dealing with emergency situations, such as natural disasters, fires,

train wrecks, civil disturbances, and civil defense emergencies. The information specialist cannot help but note the analogies between this requirement and the defense requirements of the military. The military have, of course, made large scale investments in information technology and communications, whereas the local agencies have, up to this time, tended to rely on basically manual systems.

Planning

The planning requirements of local and regional government range from broad area considerations to specific zoning variances on particular properties. The broad planning includes the general question of transportation and its interaction with commercial, industrial and residential land use. Planning for the development of undeveloped areas is one of the most important functions of a local and/or regional planning group. Maintaining the proper balance between residential and industrial usage is extremely important in the economic development of an area because it affects the attractiveness of the area, both to industry and to the work force required for industry.

Following from these needs is the assumption that a regional master plan is essential. This implies a monitoring system of existing land use and other conditions, and a measurement of the conformance of the existing conditions to the proposed plans and/or zoning regulations. This leads to more detailed questions on smaller areas, particularly the question of urban rehabilitation or renewal. Of concern here are the identification of trends toward deterioration in a specific local area and the problem of administering zoning regulations and variances for individual parcels. All of these planning functions obviously imply a sophisticated information system on the characteristics of public and private facilities in a local area.

Administration and Fiscal

The administration of a local government includes the proper accounting for tax revenues and outlays, the preparation of budgets, including the budget analysis and control function in support of the policy-making officials, the maintenance of required records by the city or county clerks, the assessment of taxes, and the treasurer and controller functions typical of any large organization. In addition to the functional needs for information in government ad-

ministration itself, there is a further provision for serving the public with respect to certain information requirements. Such data as land boundaries, and ownership records, must be available to the public on demand.

Policy-Making

Policy-making officials include mayors, city managers and legislative bodies such as city councils or county supervisors. The executive officials have typical top management functions to perform, including overall planning, resource allocation, and monitoring and control of functional activities. In so doing, they need summarizations of specific data, predictions of population and economic trends. Occasionally, they also need certain facts to counter or support criticisms from constituents of particular behavior on the part of the government. Generally speaking, these are not standard or fixed requirements but vary rapidly over time.

The legislative policy bodies have need for similar information. In particular, they require an accurate insight into the historical development of a given issue. Thus, in some cities, the actions of the city council are recorded and maintained by the city clerk in various types of file systems.

Other Agencies

In addition to the agencies that are directly a part of the city or county government, there are certain agencies concerned with local government problems who maintain field offices in the local areas. These include such organizations as the state board of equalization, dealing with the equalization of tax assessments; federal agencies, such as the Urban Renewal Administration, or the Office of Economic Opportunity; and certain commercial agencies, such as land title companies, who have the responsibility of insuring title on particular parcels in the local area. Most of these groups utilize information from local government sources. However, some maintain their own systems and have specialized requirements for which the local governments cannot and do not respond.

EXAMPLES OF COMPUTER USAGE IN LOCAL AND REGIONAL GOVERNMENT

The introduction of computers into city and county government followed closely along the pat-

tern established in the early business use of computers. Agencies which had introduced tabulating equipment for large-scale accounting and billing operations were urged by machine manufacturers to make the transfer directly to computer processing. Thus, it is typical to find the computer in most cities under the administrative authority of the controller. Types of applications for the medium- to large-scale cities and counties include payroll, appropriation accounting for revenues and disbursements, utility billing, business tax billing, and, more recently, tax assessment mailings.

Another branch of computer applications developed quite early in making the transfer from university and government uses of computers for scientific calculations to the requirements of engineering departments of large municipal agencies. Thus, it is not unusual to discover that in a particular local government, the engineering department was an early computer user.

Most of the cities' business-type applications have tended to utilize the computer in a scheduled, sequenced manner, with large quantities of data being repetitively processed on punched cards or magnetic tape. Furthermore, the introduction of computers into municipal government was usually limited to fairly large jurisdictions which could demonstrate volume and input rates that appeared to justify the computer on a purely economic basis.

Some smaller jurisdictions began, at the same time, to contract with service bureau tabulating systems, and the service bureaus eventually found it economically feasible to phase their operations into a card-processing small computer. Thus, it is not unusual for a small city, of perhaps 50,000 population, to still maintain particular applications through a service bureau.

Take utility billing, for example. The contractor takes all the input data (e.g., cash payments, utility consumption, etc.) and converts it to card form, processing the debits and credits to each account and producing the bills for mailing, as well as providing reconciliation statements for the city's financial officer. Services of this type are usually billed on a special rate for each application. A city of this size may have several applications totaling, perhaps, \$1,500 per month of contract services, supplemented by the staff personnel required by the small city to maintain the operation.

The larger cities, of course, gain certain economies of scale by having sufficient applications and

volumes to justify their own machine. Depending on the size of the city and its computer installation, they may spend between \$5,000 and \$20,000 per month on equipment rental and between 50 and 100% of that amount for operating staff and programming.

It is emphasized here that the amounts budgeted for data processing are very carefully reviewed by the policy-making bodies, and each addition of capital or staff must usually be justified on a purely cost basis. Although this is not unique to the development of new applications and markets, the particular proximity of the policy body to the operational agency in local government tends to produce an oversensitivity to this cost problem based, primarily, on political grounds. That is, in asking himself if he should vote for a particular new appropriation, a councilman must always consider whether some error, to be investigated at a later date, will cost him his political office. There does not exist the insulation of many layers of intermediate management which protect the federal policymaker somewhat in this regard and, to a lesser extent, the state policymaker.

We have described thus far the early introduction of computers at the local level. More recently, a combination of (1) technological and (2) intergovernmental developments have produced several new areas of interest and resulted in specific demonstration programs.

Technologically, the computers have become more sophisticated and powerful while, at the same time, exhibiting a marked decrease in cost. In particular, the decrease in cost of random-access auxiliary memory, and the availability of inexpensive remote terminal equipment and data communications have led to some new possibilities at the local and regional level. Coupled with this have been the recent developments in software. In particular, the availability of high-level compilers and executive systems, and the projected availability of generalized data management tools, have strongly influenced the interest in computer systems on the part of less mathematically sophisticated users. It has also made the job of indoctrination and demonstration for policy-makers easier.

The developments in intergovernmental relations stem primarily from a growing awareness at the federal level of the increasing metropolitan problems, and local governments' financial inability to deal with those problems. Because of budgetary limita-

tions there is very little opportunity for research and development financed by local and regional governments. Even the state governments have difficulty selling such nonoperational activities to the policy bodies. Agencies such as the Bureau of Public Roads, the Urban Renewal Administration, and other branches of the old Housing and Home Finance Agency, the Department of Labor, and the newly formed Housing and Urban Affairs Department (which incorporates several of the agencies previously noted), all have programs bearing on the problems of local and regional government. In the last four to five years, there has been a growing awareness in many of these federal agencies that information problems have needed to be solved before many of the substantive problems of the urban area could be attacked. Federal legislation was provided under which various demonstration programs could be initiated at the local level. We will briefly describe some of these programs below.

Metropolitan Information Systems

One of the early projects sponsored by the Urban Renewal Administration was the Metropolitan Data Center (MDC).³ Supported by a demonstration grant, the project had the task of testing the feasibility of electronic data processing equipment in establishing a Metropolitan Data Center. This Center would provide for storage and analysis of information concerning land use, housing conditions and occupancy, and related environmental factors. An objective of such a center is to maintain the information on a current basis so that it is readily available to assist local agencies in making urban planning and urban renewal decisions.

Five local government agencies were involved in the joint project from five different states: The Planning Office of Denver, Colorado; the City of Fort Worth, Texas; the Metropolitan Area Planning Commission of Pulaski County (Little Rock), Arkansas; the Tulsa Metropolitan Area Planning Commission; and the Wichita-Sedgwick County Metropolitan Area Planning Commission. The Project was coordinated by a central staff located in Tulsa. Separate centers in each city provided data processing services for the local planning agency. Interagency arrangements and central staff were employed only in relation to project development.

Applications developed included a comprehensive land use plan for Denver; the capital improvement program in Wichita; community renewal program in

Tulsa; a central business district plan in Fort Worth; and a school facility plan in Little Rock.

Benefiting from some experience gained by the Metropolitan Data Center Project, Alexandria, Virginia, has instituted what is called a "data bank".⁴ A master file has been created on magnetic tape of every parcel of land in the city, approximately 20,000 parcels. Each record contains the information concerning the parcel, land use and space use. The data bank is utilized by every major department and agency in the city, although it is a relatively small operation in a batch process form.

Another major effort of this type was undertaken in the City of Pittsburgh, some time earlier. A computer-processed land record system was developed which provides retrieval of various items concerning parcels in the city.

Another significant early development was the PENJERDEL study.⁵ This was a joint Pennsylvania, New Jersey and Delaware Metropolitan Data Study, which was exploring feasibility rather than actual development. A contract with the University of Pennsylvania resulted in an area statistical project. Five types of area data service were to include a data utilization center, a land use and parcel inventory, traffic information, regional accounts, and capital expenditure evaluation.

The concept of a metropolitan area information system is growing in acceptance, but as yet there is no system on a broad area basis that one might call operational. The feasibility of the concept is being explored in several places across the country. In the City of Los Angeles, a study was recently completed by System Development Corporation of an Automated Planning and Operations File, called APOF.⁶ The conceptualization of the APOF system included a basic random access file of all 900,000 parcels in the entire city of Los Angeles. The information would be accessible to all city departments and would include data by both city and noncity agencies. This is undoubtedly one of the largest systems ever conceived for a metropolitan area. A specific plan for implementation of that system was presented in the study, and methods of financing are now being considered.

Another major project in the Southern California area is funded with a federal grant. This is the South Gate Municipal Management Information System (SOGAMIS).⁷ This project has the twofold purpose of developing a computer management program for South Gate, California, a city of slightly over 58,000

population, and to prepare a model plan for any city between 30,000 and 300,000. The project is being conducted by the University of Southern California. Specific goals of the project are: master zoning plans based on population density and new construction; scheduling of police patrols, using information of types and frequency of crime throughout the city; planning new fire stations and optimal distribution of equipment; handling traffic problems, using accident flow and accident patterns; and planning of school, park and playground sites. This project is now in the early development stage.

Operational City Systems

Some operational systems of a more modern variety may be seen in several cities and counties across the country. In the Bay Area of California, the Alameda County Computer Center provides a random-access system for welfare information for its various social agency departments. Alameda County is also providing real-time retrieval of warrant information for several law enforcement agencies in the Bay Area.

Other police systems of an operational support nature exist in the cities of St. Louis, Chicago and New York. In Los Angeles, large scale experimentation was undertaken in the Police Department on the use of computers to process crime information in natural language.⁸ Incorporating those concepts, a system design study⁹ was recently completed by System Development Corporation recommending an operational system to process crime and arrest data, providing outputs to field patrol and investigation, as well as providing management information.

One of the most interesting recent developments is a proposal by 19 cities in the San Gabriel Valley, Los Angeles County, to cooperate on a central data service for the entire Valley. A feasibility study is under way as of August 15, 1966, and results should be available for discussion at the AFIPS meeting in November. It is expected that the feasibility study will concentrate on certain "bread and butter" applications, such as utility billing, appropriation accounting, payroll, and police statistics. More complicated functions such as land use records and managerial and policy decision support will also be considered. The study will examine several alternative ways of providing a central service, including the possibility of a central time-shared facility, either operated by a central agency or provided by some

service bureau. Methods of cost-sharing, scheduling and priorities will also be derived so that each city can expect the service to meet its functional requirements.

Area Transportation Studies

Some of the largest regional planning functions have been accomplished under the aegis of area transportation studies. Significant projects have included the Chicago Area Transportation Study; the Tri-State Transportation Study, covering New York, New Jersey, and Connecticut; the Penn-Jersey Transportation Study;¹⁰ and more recently the Bay Area Transportation Study¹¹ in California. In all of these efforts, large field data collection activities were undertaken concerning the characteristics of commuter travel and projected trip requirements. Models of regional economic growth were constructed and the data used to project effects of alternative transportation facilities. As might be expected, it was usually found that the interaction between the transportation planning and the land use and economic planning was quite significant. As a result, it has become increasingly clear that one of the functions regional organization must provide is a mechanism for maintaining current area-wide information. Periodic studies are not adequate; the data base must be continually updated if planning information is to be at all valuable. The information system implications of this approach are rather significant.

SOME PROJECTIONS FOR THE FUTURE

There is no question but that the population in the urban areas will continue to increase. The service functions of government will even more strongly reflect this trend. This means that the information generated and processed in local and regional government will increase in volume at an even higher rate. When we consider the increasing labor rates, and the decreasing cost per unit of information processed by computers, it seems clear that local and regional government will provide ever-increasing opportunities to introduce advanced technology. The question is, how can the local governments overcome the budgetary and political limitations that tend to retard the introduction of new concepts?

One trend that we believe is unmistakable is the movement toward sharing of central computer facilities. So far, this has mostly been exhibited in the administrative functions, based on the argument that

centralization of data processing brings economies in government. In the past, the decentralized user has usually suffered in service when centralization has taken place. This is why the individual city agencies that have their own machines fight centralization so energetically, and yet, so unsuccessfully. It is certainly true that by centralizing facilities, and systems and programming staff, the consistency of programming is improved and the fixed investment is minimized. However, the result of this approach has usually been to frustrate individual departments and agencies who wish to experiment with new approaches, or who have special requirements that are not served as well as if they operated their own facility. How can we solve this dilemma?

We believe that the trend will continue toward centralization of *facilities*, so that the city can achieve the economies of scale available. However, the difference in the future will be the decentralization of *input and output* of data, plus the time-sharing concept of *different users operating their own programs*, built with generalized data management systems and high level compilers. Certainly some efficiency of operation on repetitive functions will be sacrificed, but the overriding consideration will be the availability of computer power to a much wider range of users and applications than is presently possible with central programming staffs and equipment.

In the larger metropolitan governments, this will mean that more operational, planning, and administrative functions will be serviced. In the smaller cities, the time-shared concept will mean that jurisdictions who cannot afford large powerful computers by themselves will be able to get the processing advantages of such systems, with expenditures equal to or less than what they currently spend. This will be accomplished either through service bureaus providing time-shared capability, or through the kind of regional arrangements implicit in the San Gabriel Valley project.

From the functional and political standpoints, this trend will have many salutary effects. The present resistance of individual jurisdictions to join in regional associations is based primarily on the fear that centralized government would result. By decentralizing the user of regional information through the kind of network implied in the time-shared facility, the threat of centralized *control* is reduced, yet the value of *information sharing* for planning purposes is retained. Furthermore, specific functional areas

such as the administration of justice, or area transportation, can be served by specific subsystems devoted to that function alone. Those subsystems would be tied to the overall regional or statewide network for purposes of sharing nonsensitive data.

Although these long-range goals are certainly desirable, and although we do believe them to be inevitable (given the present movement of technology and government), there are some serious problems in making a smooth transition from the present state. For one thing, the process of introducing innovation in government is fraught with costly discontinuities in funding, and nontechnical influences on content which severely constrain any real progress. This is partly the fault of the information specialists, who have tended to ignore the requirement for communicating effectively to nontechnical executives and policy officials. The information specialist must take some pains to learn other languages besides PL-1, COBOL and FORTRAN, and deal with political and administrative loops as well as those he normally encounters.

Finally, there is the problem of software development. The local governments have a serious problem, undoubtedly of their own making, in attracting and keeping trained programmers. It is not even clear that raising present salaries would be effective, because industry tends to keep ahead of government in this area, and can do so more easily because they do not suffer from the same legislative time lag. As a result, the local governments cannot maintain continuity of staff. This produces error-prone application programming, or just plain schedule slippage. Contracting for software isn't a solution for two reasons. First, because the higher cost per man-hour of contract services reduces the total output possible for the same dollar (assuming a well-paid internal staff could produce the same output), and, second, because the maintenance function must be accomplished internally anyway. This means that the internal staff must be involved somehow in the initial program preparation in order to fully understand what is going on.

The long-range answer to the software problem in government lies, we believe, in the development of

generalized program systems, either specific applications (e.g., water billing, payroll, statistical processing) or data management tools that can be used for general information storage, retrieval, analysis and reporting. The funding of such developments must come from outside the local or regional governments, and probably from outside the states. Only with the advent of these types of systems will the impact of computers on local and regional government reach the potential possible.

REFERENCES

1. Lockheed Missiles and Space Co., "California Statewide Information System Study—Final Report," July 30, 1965.
2. H. H. Isaacs, "The Metropolitan Problem: Some Facts in Summary," System Development Corp. SP-1941 (Jan. 20, 1965).
3. *MDC Project Report*, Feb. 1966.
4. John K. Parker, "Operating a City Databank," *Public Automation*, June 1965.
5. W. Alderson and S. J. Shapiro, "A Metropolitan Data Bank for the Business Community," *Business Horizons*, summer 1963.
6. H. H. Isaacs, W. O. Crossley, and D. R. Pascale, "Final Report of the APOF Conceptualization Study," System Development Corp. TM-(L)-2905 (Mar. 31, 1966).
7. W. H. Mitchel, "SOGAMIS—A System Approach to City Administration," *Public Automation*, Apr. 1966.
8. H. H. Isaacs and W. W. Herrmann, "A Computer-Based System for Processing Crime Information," *Industrial Security*, Feb. 1966.
9. L. B. McCabe et al, "Los Angeles Police Department Phase I Operating System Description," System Development Corp. TM-(L)-2506 (Dec. 31, 1965).
10. R. M. Zettel and R. R. Carroll, "Summary Review of Major Metropolitan Area Transportation Studies in the U.S.," University of California, Berkeley, Institute of Transportation and Traffic Engineering, Special Report, Nov. 1962.
11. Bay Area Transportation Committee, Progress Report, Mar. 1, 1965.

AN INFORMATION SYSTEM FOR LAW ENFORCEMENT

LeRoy B. McCabe and Leonard Farr

*System Development Corporation, Santa Monica,
California*

INTRODUCTION

The System Development Corporation (SDC), in conjunction with the Los Angeles Police Department (LAPD), conducted a System Design Study for Phase I of the LAPD Information System during the period of June, 1965, to January, 1966. The System Design Study was organized into two major activities—a System Analysis and a Phase I System Design. Each of these activities was further divided into major tasks which were performed by SDC and LAPD personnel working in close harmony.

The major product of the Systems Design Study was the documentation of a concept for the design and operation of an automated information system. This concept is detailed in the Operating System Description¹ for Phase I of the LAPD System. In support of this concept, other documented products resulting from the Study were the System Analysis, the Equipment Specifications,² a System Development Plan,³ and a Glossary⁴ of terms used in law enforcement and information processing. The evolutionary approach to system development was adopted; i.e., the System is to be designed and built in blocks or phases concentrating first on the most pressing operational needs.

The Phase I System was limited at the outset of the Study to the following applications:

Crime and related reports.
Want and warrant information.
Field Interview information.

However, it was also possible for the designers to consider certain aspects of the arrest, booking, and jailing operations. Other critical operations such as personnel deployment and traffic enforcement were deferred to a Phase II effort.

This paper summarizes the results of the analysis and design activities by describing the most salient features of each. It discusses the implications for the Phase I System as a regional system and provides directions in which the LAPD System can expand to develop, ultimately, into a total law enforcement command and control system.

SYSTEM ANALYSIS

A System Analysis, ideally, is performed on the "entire" system and not a portion of the system. If allowed to proceed ideally, the analysis would encompass every aspect of a law enforcement operation. In adhering to the evolutionary approach, however, the System Analysis concentrated primarily on those Department operations within the bounds of the Phase I System, but also considered, in general, other operations with a view towards future applications.

The activities and documentation of the results of the System Analysis* were divided into four parts: a Configuration Analysis, a Report Forms Analysis, a Functional Analysis, and a Requirements Analysis. This documentation is quite extensive (some 800 pages) and is intended to establish a detailed and comprehensive understanding of the present LAPD information processing procedures. Understanding of current operations is essential to the success of subsequent design tasks.

All of these analyses were conducted jointly by SDC and LAPD personnel under the guidance and direction of SDC personnel. In preparation for each of the four separate analyses, a "Guide" was published for the benefit of LAPD personnel to acquaint them with the objectives and procedures involved in that particular analysis. These Guides and a three-day orientation period constituted the only formal training of LAPD personnel prior to their participation in the analysis activity.

Current LAPD Operations

The operations of the Los Angeles Police Department are typical of those of a large metropolitan department. The Department is organized primarily in decentralized field commands, housed in buildings located throughout the entire city. Responsibility for patrol effectiveness against the crime problem rests with the field division commander. Not all of his responsibilities, however, are confined to patrolling against known patterns of activity. Many of the actions of the individual patrol unit occur in response to requests for service through the central communications system. When an event occurs, in most cases, the central communications system receives a telephone call from a citizen. The officer receiving the telephone call decides that some action on the part of a field unit is necessary and a dispatcher selects the appropriate field unit and relays the call number and message to a radiotelephone operator.

The field officer often requests information on individuals or vehicles which he needs for his immediate decisions on action to be taken.* The central

* Los Angeles Police Department System Analysis, TM(L)-2497, Volumes 0 through 4, System Development Corporation, December 31, 1965. This document is releasable only through the Los Angeles Police Department.

* For example, in the City of Los Angeles alone there are over 600,000 inquiries from field officers each year relevant to possible stolen vehicles, with an additional 600,000 inquiries from field personnel relevant to individuals for whom warrants of arrest may be outstanding.

communications facility must be in constant contact with the information files of the Department. These files include both the kinds of information required in real time, such as warrants or wants for individuals; and the type of information with less critical time requirements, such as criminal records, crime reports, and investigator's follow-up reports.

The crime reporting process usually begins with the field officer. When an event occurs and the field unit arrives at the scene, the field officer will make a report of the crime. That report is then reviewed by his supervisor in the field division headquarters. The crime report becomes the basic information upon which detective follow-up is based. It is also used to prepare statistics for command and management purposes.

Investigative activities with respect to crime are also generally decentralized at the division level with the exception of a number of specialized details such as narcotics and abortion. Investigative activities involve the patrol, supervisory, and detective force in a significant information processing effort, especially to support crime pattern recognition. When a new crime occurs, the patrol commander would like to know if this crime is linked to previous crimes in his area or if, in fact, it is related to crimes that have occurred outside his area of cognizance. Patterns are maintained at the divisions by analytical officers with the aid of pin maps.

The investigator relies heavily on the crime reporting process, especially when he is attempting to link up a series of cases in order to develop additional leads or suspects. When a suspect is apprehended for one particular offense, the investigator needs to be able to search his file for uncleared crimes that the suspect may possibly have committed. All these activities depend on an effective information system that can aid the processes of retrieval, analysis, correlation and dissemination.

Current LAPD Operational Problems

The Los Angeles Police Department, as do most metropolitan police departments, faces crime loads which are increasing at a faster rate than population in the urban areas. For example, during the period from 1954 to 1964, the City of Los Angeles experienced an increase in population of slightly over 23 percent while the total of all crimes reported to the Police Department, during the same period, increased over 120 percent.

The growing operational load carries with it a significant requirement for processing of information concerning each event. The costs of the processing required to keep up this volume generally far exceed the limited financial resources of the metropolitan community. The inevitable result is that many types of information that would otherwise have operational value are not readily accessible to the Department.

The problems of sheer volume of information to be processed do not adequately describe the scope of the problems facing police departments. The patterns of crime have become more complex in type of event, modus operandi of the criminal, and the difficulties in matching a suspect with a given crime.

The greater mobility of the criminal, brought on by the increase in motor vehicles and freeways, has had serious impact on the nature of field operations. It is no longer possible to concentrate crime pattern recognition in a given field division. The criminal does not respect divisional boundaries. In fact, it is no longer feasible to attempt crime pattern analysis purely within the bounds of any single political entity. An obvious corollary of this problem is the effect on interagency communications and information processing within a given region. The police departments interacting with the Los Angeles Police Department need to have access to information contained in LAPD files, since they use those files for much of their information support. Similarly, the Los Angeles Police Department must have immediate access to information collected in neighboring police departments and in its large sister agency, the Los Angeles County Sheriff's Department. This requirement for interagency cooperation has been considered in the system design activity and is discussed later.

The general problems described above lead to specific needs of the Los Angeles Police Department. First, with respect to the processing of crime information, the essentially manual information handling techniques presently employed encourage duplication of files in an attempt to minimize information response time and do not contribute to effective correlation and retrieval of stored information. When compared to available electronic data processing capabilities, these manual techniques are outdated and stressed to a point that precludes accommodation of any increase in demands.

The general deployment of patrol forces is based on outdated statistics, weighing factors, and analytical techniques, many of which, due to lack of availa-

ble advanced technological means, have not been adequately tested. It is highly desirable to deploy forces by means of immediate analysis of statistics that are current at the moment of deployment. This cannot be done utilizing present techniques, equipment and procedures.

Although these problems are expressed here in terms of the Los Angeles Police Department, they are also typical of every large metropolitan law enforcement agency and representative of problems encountered in those suburban areas which depend on the aggregation of services of many small police agencies.

SYSTEM DESIGN

Background and Objectives

The law enforcement needs in Los Angeles, especially with respect to information problems related to crime, were recognized several years ago. In 1963, the City undertook jointly with System Development Corporation a program of research and experimentation in the application of advanced computer techniques to the processing of crime data. The approach undertaken at that time emphasized "natural language processing." Rather than extend the errors and ambiguities of numerically coded crime data (as currently processed on punched cards), the computer was used experimentally to process and retrieve crime report information in its original English language form. A research and development program, conducted over an eighteen-month period, validated the concept that the computer could assist the investigator in crime pattern recognition, using this advanced approach.

It was also recognized that a future operational system for the LAPD and/or a regional law enforcement complex would envision inputs and inquiries to a central area-wide file from remote stations in field locations. Therefore, in addition to exploring natural language retrieval, this early SDC-LAPD Project also tested the concept of computer time-sharing. The term "time-sharing" is used here to denote the concept of the simultaneous use of a central computer by multiple users, each operating a different computer program, communicating data and instructions to the machine from remote locations, and receiving on-line responses. Each user has his own input/output device such as a teletypewriter or graphic display on television-type cathode ray tubes.

In June, 1965, after this early experimentation the City of Los Angeles contracted with System Development Corporation to undertake a design effort, applying these concepts and research results to the first phase of an information system to serve the Los Angeles Police Department. This system design effort concluded in January, 1966.

A summary statement of the initial objectives of the system design effort, in light of the problems previously outlined, includes:

- (1) Achievement of more financially economical means of processing increasingly larger amounts of information presently associated with crime and related reports, wants and warrants, and field interviews.
- (2) Minimizing the time required to process and communicate the information relevant to crime and related reports, wants and warrants, and field interviews.
- (3) Maximizing the accuracy, relevancy, availability and effective utilization of crime, wants and warrants, and field interview information.

The Phase I System is described below in terms of the inputs, processing, and outputs available to system users. However, the following are some of the System's key characteristics:

- (1) The System is based on the operation of a general purpose digital computer.
- (2) Information input to the computer will be entered by telephone through a centralized conversion pool.
- (3) All information will be reviewed first on a division level, and then by a command inspection activity that will provide centralized control over inputs to the System.
- (4) Access to the System will be time-shared by means of keyed display devices.

Phase I System Applications

Figure 1 and the following discussion summarize the applications encompassed by the Phase I System:

- (1) The processing of crime data on a City-wide basis, including field report-

ing of events in remote locations; entry into a central computer file for correlation with similar data; dissemination of abstracts back to the field location both automatically and upon demand; the production of the management reports to be used for evaluation and deployment of manpower; assistance to the investigator in retrieving relevant crime data; and the automatic dissemination of information to concerned county and state agencies about the crime activities, stolen property, individuals and vehicles involved.

- (2) The maintenance of a real-time inquiry system for warrants and other wanted individuals.
- (3) A total inventory and processing of arrest and booking information, maintaining up-to-date information on individuals who have been arrested by the Los Angeles Police Department. This system will be compatible and communicate directly with the arrest and jail system of the Los Angeles County Sheriff.
- (4) The incorporation of data on persons contacted in the field and the correlation of these data with possible crime occurrences in the same area. These field interview data are currently collected by patrol officers throughout the City, but have limited application because of the difficulty of correlating and retrieving information from the manual system. With the new system, these data would be available, on either automatic or request basis, for computer retrieval and matching with selected crimes. This will be accomplished within the system by a series of computer programs referred to as PATRIC (Pattern Recognition and Information Correlation). As information from the various System inputs is stored in the PATRIC Index, the PATRIC retrieval and correlation function will automatically operate upon, correlate and output those particularly similar events that

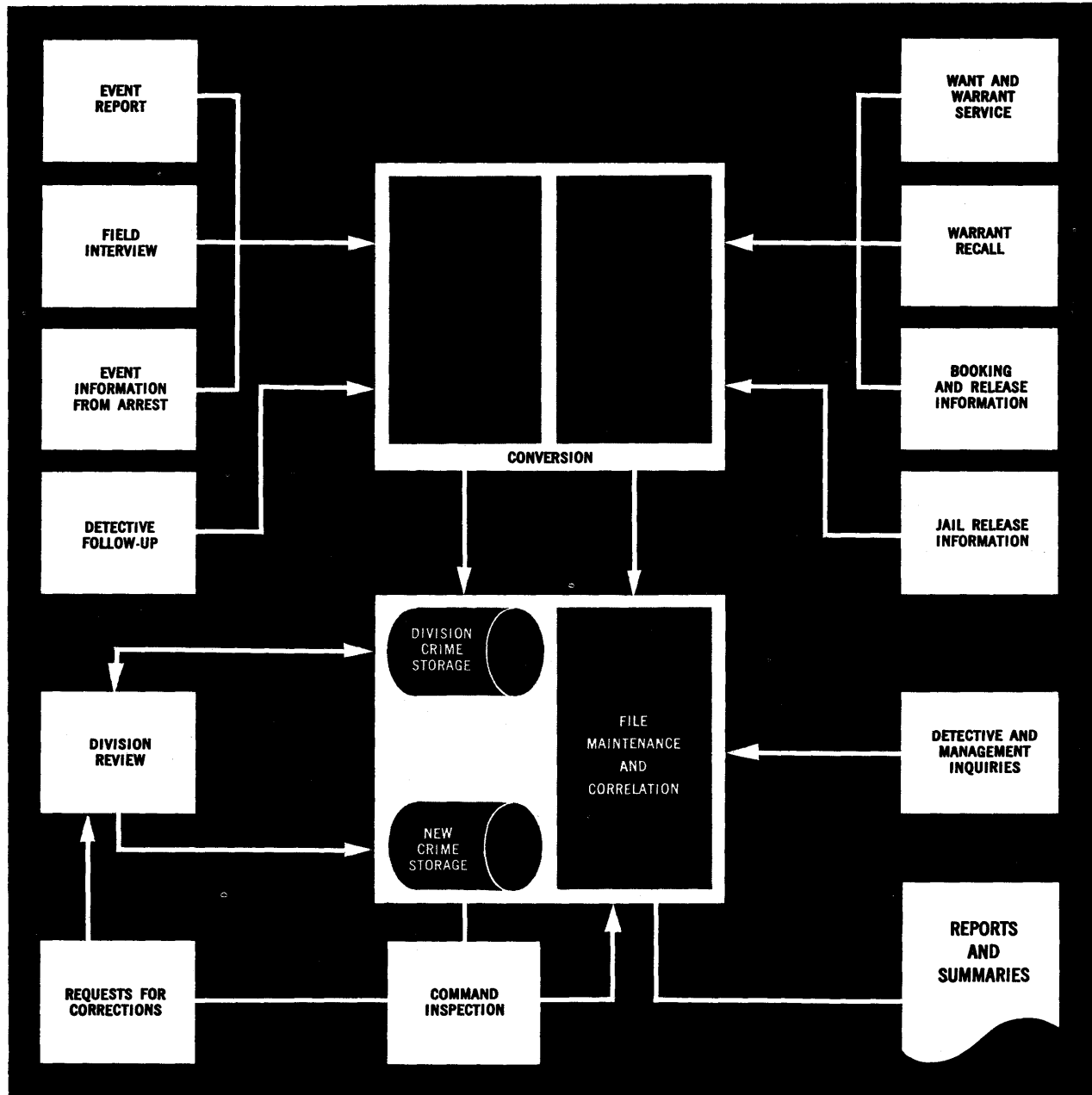


Figure 1. LAPD System Phase I applications and processing.

appear to be related. The results can be dispatched automatically to the concerned investigators at the beginning of each day for their evaluation.

Inputs, Conversion and Review

The primary information accepted by the Phase I System will include:

- (1) Event Reports and related follow-up reports (all crime information).
- (2) Contact information derived from field interviews and arrests.
- (3) Booking and release information within the City Jail System.
- (4) Want and warrant information derived from the receipt, registration and processing by the Department of

all parking warrants, felony and misdemeanor warrants, and felony wants. Included in this category are all local and other agency wants and warrants. In all instances, however, a copy of the warrant must be physically in the possession of the Department or a reasonable justification for a want from another agency must be present.

With the exception of hard-copy want and warrant information, the primary mode of reporting will be by telephone to the Conversion Center. A Cue Sheet will be used by the reporting officer to assist him in organizing and specifically noting information collected at the scene of an event, during a contact (field interview or arrest), or in the booking process. The objective of the Cue Sheet is to systemize and reduce the amount of reporting without degrading the quality of information.

Information reported by telephone will be recorded at the Conversion Center in the Department by a device controllable from any telephone. Hard-copy information (e.g., wants) will be converted directly. Event and Contact Reports will be converted without delay by transcription personnel situated at keyed display consoles in the Conversion Center. The use of the keyed display consoles will allow conversion personnel to edit the data (for conversion and format errors) prior to the insertion of the data into the appropriate System storage area. Corrections and other similar modifications will be made at the console. The reports are then passed by the System to the Division Reviewer for correction or modification, and approval prior to their entry and availability in the System. Arrest and crime reports (i.e., event reports) concerning individuals and crimes in the City but prepared by other law enforcement agencies will be processed into the System in the same manner as LAPD-collected information.

Booking information will be transmitted by either teletype or telephone at the point of booking to the central transcription pool in the Conversion Center. Booking and release information will be converted immediately for dissemination. The System will respond by printing the booking (or release) information at the appropriate points (division of booking, Central Jail) on the Department Teletype system in a sufficient number of copies for the booking process. Multiple copies may be produced on formsets, depending on the eventual use of the printed book-

ing report. Figure 2 depicts the major equipment components in the Phase I system.

Information concerning the service or attempted service of warrants will also be inserted into the System. If a warrant has actually been served, this action will be reflected through the booking process. If an attempt has been made to serve a warrant but it was not served, the reasons for non-service will also be entered into the System. As each attempt is made, the reason for non-service and the number of attempts will be recorded in the want and warrant file.

As Event and follow-up reports are reviewed and approved at the field division level, they will be automatically displayed (by type of crime) on one or more of the keyed display consoles in the Command Inspection area (Figs. 1 and 2). The report of each type of crime is then inspected by Command Inspection specialists who will make appropriate corrections or modifications and request additional information from the concerned divisional personnel when necessary. Command Inspection specialists will review the report contents to derive the appropriate MO when necessary, make investigation assignments when required, evaluate the completeness of information, and assign the principal crime in those Event Reports describing multiple crimes. When satisfied that all Event Report requirements have been met, Command Inspection will release the information into the System.

Outputs for Line and Management

The retrieval function will provide the user with a means for searching the total collection of crime and related information contained in the system and present him with organized responses. The correlation of information, such as peoples' names and descriptions, locations of crimes, descriptions of vehicles and methods of operation, will be a prime capability of the retrieval function. A copy of any specific report based on its identification number can also be obtained by the user if he desires.

For example, if a user is looking for a certain type of crime report, he might include in his request a description of a suspect (with or without a specific name), the specific MO known to be associated with the suspect and, perhaps, a partial vehicle description. The computer would then search its total data base and refer to all reports (whether derived from

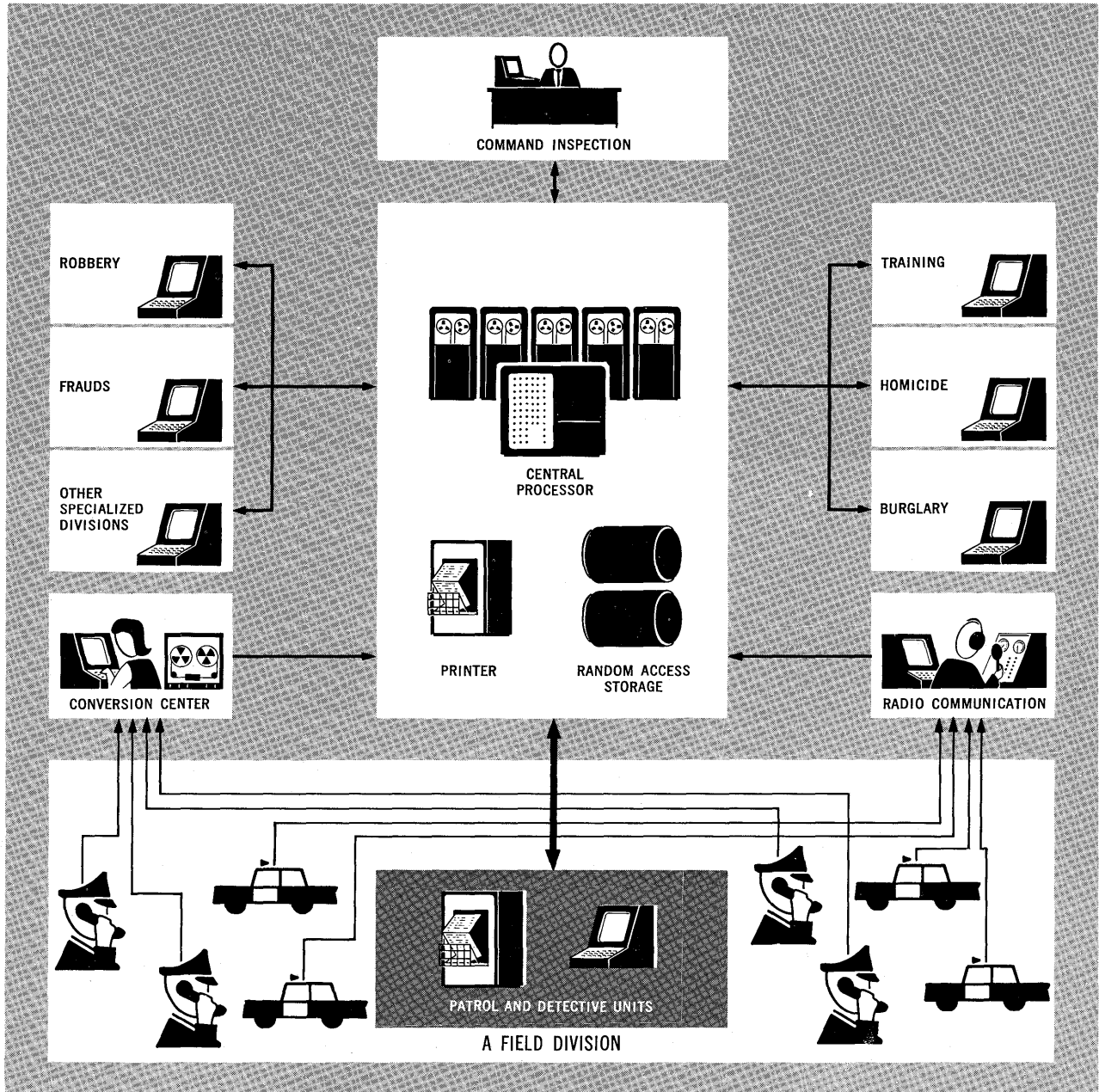


Figure 2. LAPD System Phase I equipment configuration.

events or field interviews or from other reports) that contain any of the information the user has requested.

Another capability of the retrieval function will be the automatic correlation of information in older reports with that included in selected new event reports. In addition, if desired by a particular investigating officer, a check can be made against contact reports based on time and location (as well as sus-

pect description) to determine if any field interviews were conducted at approximately the same time and in the same general location. Thus, each investigating officer can be provided with all possible relevant information on the crime or crimes he is investigating.

The overall retrieval function, including both the automatic and special request modes, is referred to as **PATRIC** (Pattern Recognition and Information

Correlation). The following hypothetical situation is an example of the PATRIC operation:

- A "window smash" burglary occurs at 1:00 a.m. at Wilshire and Fairfax. Witnesses describe the perpetrator as a male caucasian, 5'9" tall, wearing light clothing. The field officer completes the report and the information is entered into the System.
- Approximately an hour later at 2:00 a.m. at Westwood and Pico, in an adjacent division, a person fitting the same description is stopped and interviewed by other field officers. A Contact Report (field interview) is completed and entered into the System. The additional name and address information is now on file.
- At 2:30 a.m. the same suspect, in the Venice Division, is arrested and booked for being intoxicated. His name, physical description and other pertinent data are inserted into the System.

From the time the Event Report entered the System, PATRIC has been automatically correlating it with other similar events in the System files. The results will arrive on the concerned investigator's desk the next day and may include the following correlated information:

- (1) A window smash occurred at 1:00 a.m. at Wilshire and Fairfax; the suspect was a male caucasian, 5'9" tall, wearing light clothing;
- (2) At 2:00 a.m., at Westwood and Pico, a male caucasian, 5'9" tall, wearing light clothing, and named John A. Doe was interviewed and released;
- (3) The Warrant File has been automatically searched, and there is no want or warrant on file in that name for the stated date of birth;
- (4) John A. Doe has had several Contact Reports made on him at various locations within the last three months;
- (5) John A. Doe has been arrested and booked before;
- (6) All these events took place within a 2 to 3 hour time period and within a radius of 10 miles from the initial event;

- (7) A John A. Doe is now in custody at Venice Division jail awaiting arraignment on a drunk charge.

The rest is left to the investigator. He may make additional requests of the System for clarification of certain data or he may disregard the information.

Statistical summaries of crime activities for the City will be available periodically or on demand. Because Event Reports will be processed into the System shortly after the collection of the information, the current status of criminal activity will be available almost immediately. A major function of the Command Inspection operation is to assure the Department's current awareness of the citywide crime situation. Event Reports concerning special occurrences (such as reports involving a shooting) will be brought not only to the immediate attention of the Command Inspection personnel, but also to other Department personnel whose action is required. Special listings of abstracts or reports will be provided either in response to a standing request or by special query. By using the statistical analysis function in the System, requests can be made for special summaries or even more involved statistical analysis of information contained in the various files of the System. In most instances, statistical summaries presently prepared for or by the various divisions will be produced by the System automatically or on request. Aside from the major reports (Event and Contact) resulting from the collection of crime and related information, the System will derive and produce numerous other reports and summaries such as the Jail Population and Crime Abstracts. Information will be as current as the most recently inserted data.

THE PHASE I SYSTEM AS A REGIONAL SYSTEM

The expansion of the Phase I System for use as a regional information system for law enforcement agencies in the greater Los Angeles area is not only feasible, but reasonable. Some requirements for the implementation of such a concept include:

- (1) The need for increased random access computer storage capacity.
- (2) The requirement for at least one keyed display console or teletypewriter to be located at each participating agency. Additional input-output

equipment may be required depending on input volume and query rate.

- (3) The need to establish efficient data conversion procedures on the part of the participating agencies.
- (4) The requirement to specify the degree of interagency access to information in the data base.

The most immediate question is that of warrants and warrants. The Municipal Court and the Los Angeles County Sheriff's Department, as well as LAPD, have been considering warrant processing requirements. Additional interactions among concerned agencies will be required to determine the ultimate regional system configuration.

Provision has been made in the design for the interaction of the LAPD System with other City, County, and State information systems. When the LAPD System becomes operational, an interface will occur with the Police Information Network (PIN) of Alameda County and the AUTO STATIS (Automatic Statewide Auto Theft Information System) operation of the California Highway Patrol. Contemplated systems of other agencies, with which the LAPD System may have to interface, include the National Crime Information Center, the Department of Motor Vehicles, the Bureau of Criminal Identification and Investigation, and other agencies such as the Municipal Court and the Los Angeles County Sheriff's Department.

Regardless of whether a regional system is developed, agencies presently receiving information from the LAPD will receive that information in the form of automatically produced reports in a format acceptable to them, or if desired, in the form of magnetic tape containing the information.

FUTURE ANALYSIS, DESIGN, AND DEVELOPMENT

The ultimate goal of the joint LAPD-SDC study team is to analyze all information processing tasks now being performed in the LAPD. The evolutionary development of the system will insure the LAPD of an early operational capability that will solve some of its immediate problems while analysis and design activities continue on future phases of the system. This approach also allows for the gradual phase-out and transition from the existing manual system, thus helping to introduce features of the au-

tomated system to operating personnel. Most important, the capability for responding to new or changing operational requirements will be maintained.

Problems in implementation and operations will also be somewhat minimized by the evolutionary approach. By automating only a few elements at a time, for example, training for and transition to the new system will be eased. However, there are also some problems associated with this evolutionary development. They stem from the faulty assumption that succeeding phases are independent and can be dealt with separately when, in fact, each of the phases must be analyzed and designed with the objectives of the entire system at the forefront of the activity.

Management must select the elements to be included in each phase, basing its decision on such factors as cost, ease of implementation, resulting cost savings, increased effectiveness and general desirability, just as it did for Phase I.

Although the full range of capabilities has not as yet been determined—special studies are required before specific applications can be selected—there are a number of suggested applications to be included in succeeding phases.

It is expected that Phase II will computerize additional arrest report activities, daily field activity reports, traffic accident reports, traffic enforcement selective deployment, officer deployment and car plan generation.

A capability offering great promise—fingerprint identification, storage and retrieval—will be included in Phase III together with property reports and a pawnbroker cards file.

Finally, during Phase IV, personnel, training, motor transport, and supply and equipment records can be processed and maintained by the automated system. These non-law-enforcement applications can be considered independently of the information system described here, but they are included for the sake of completeness in the system analysis and design.

REFERENCES

1. L. B. McCabe et al (SDC) and Sgt. H. H. Ebersole et al (LAPD), "Los Angeles Police Department Phase I Operating System Description," SDC Document TM(L)-2506, December 31, 1965.
2. J. T. Wagliardo, "Los Angeles Police Department Phase I Equipment Specifications," SDC Document TM(L)-2504/000/01, December 31, 1965.

3. L. Farr, "Los Angeles Police Department System Development Plan," SDC Document TM(L)-2502/000/01, December 31, 1965.

4. SDC-LAPD Joint Study Team, "Los Angeles Police Department System Design Study Glossary," SDC Document TM(L)-2498, December 31, 1965.

THE TRANSFER OF SPACE AND COMPUTER TECHNOLOGY TO URBAN SECURITY

Richard B. Hoffman

*Center for Planning & Development Research
University of California, Berkeley*

INTRODUCTION

The case for the application of systems analysis and computer technology to the problems of urban management has been a "timely" and important subject for the past five years. The automation of administrative procedures appears to have been successful in the Chicago Police Department, where crime statistics, operations reports, traffic accident and citation reports, payroll, financial accounting, and automotive cost accounting are presently working components of the Chicago Police Department's data processing system.

A similar rather extensive effort is presently taking place in Los Angeles, but the study group's proposals are based on the existing organization, which very well may not make the best of the available computer technology.

The Chicago system, which utilizes an 80K IBM 1411 CPU with a wide range of peripheral equipment including a disk storage unit and accompanying remote inquiry units, has capabilities and applications well beyond its old system; this, however, should be attributed to the reorganization of the department accompanying the appointment of Superintendent O. W. Wilson, formerly Dean of Berkeley's School of Criminology, which allowed Chicago's computer system to be implemented in a climate of change.

These cities have sufficient resources to justify large-scale computer installations which merely automate the existing systems. The smaller public units will require either regional installations* or the broad application of systems analysis to take the greatest possible advantage of today's computer technology. This latter course is perhaps the only economically feasible alternative for the medium large city with a population of 100,000 to 300,000. However, the problems the systems analyst faced with the industrial manager in attempting to recast missions and question "basic objective" are but a small indication of the resistance to be met in working with the public manager whose natural aversion to change is reinforced by statute, code, and administrative and civil service regulations. Recent work in a San Francisco Bay Area city has particularly highlighted this problem, even though the study city does have an unusually cooperative administration.

* The financial support of the county of Alameda was necessary to implement PIN (Police Information Network) in the San Francisco Bay Area. However, the willingness of the Supervisors of Alameda County to support PIN was based upon the system's eventual ability to be self-supporting by performing adequately a task which was being performed inadequately. That, in fact, some important costs, namely, that of bringing the offender (individual with open warrants) into jail, were neglected is an indication of the existing "systems approach." County of Alameda—1964.

The other side of the problem is reflected in aberrations in performance to control standards. The original intention of any set of controls is to assure that the organizational objectives are fulfilled. However, most operating controls reflect only a portion of any multiple-objective criterion function* and therefore we can expect to find numerous instances of aberration¹ and suboptimization.² However, we must caution those who seek to use naively the computer system to strengthen the existing control system to reduce the "aberrant effects." They should remember that the controls are only an approximation of the firm's objectives. And, if controls are followed blindly, whether through a lack of awareness, coercion, or for personal gain, the effect upon organizational performance can be equally detrimental.

It appears that the problem of implementing an effective systems analysis into the urban management context will be two-fold; (1) to obtain cooperation for a full, rather than a piece-meal systems effort; and (2) to devise and communicate a control system that adequately reflects organization objectives.

It is necessary to first look at the primary missions of the security function to determine what the police department organization believes it is doing, so that it is then possible to evaluate how well the department is performing against its announced objectives. Happily, there appears to be considerable agreement among writers in the field of law enforcement, at least at this level.

Table I. Function (Missions) of the Law Enforcement Agency

1. Prevention	1. Crime Prevention	1. Prevention
2. Repression	2. Patrol	2. Surveillance
3. Arrest, recovery of stolen property and preparation of cases	3. Detective and Vice	3. Apprehension
4. Regulation of people in non-criminal activities (traffic, crowds) ³	4. Traffic ⁴	4. Traffic ⁵

* Even if the control system is designed to measure performance to each of the multiple-objectives, there exists the problem of determining the weighting (mapping) of the individual objectives and the communication of this weighting to the operating group.

There exists the problem of determining which of these areas will yield the greatest possible pay-off for a given level of effort in applying systems analysis. Although it is realized that the entire urban system must be studied to yield the maximum benefit from the analysis, separation into the various sub-functions can yield tangible results provided one is willing and capable of evaluating the effects of the functional interrelationships when these effects are "critically" relevant.

APPLICATION OF SPACE AND DEFENSE DEVELOPED TECHNOLOGY TO THE PRIMARY MISSIONS

Although each of the primary missions will be evaluated separately, this approach does not in itself preclude technological applications which pertain to more than one of the primary missions, or which have interrelationships with parts of the urban system other than the security function.

Traffic

The problem of traffic or the regulation of the public in non-criminal activities can be viewed as the regulation of the flow of vehicles and pedestrians. It would then appear that the traffic mission would best be performed by minimizing the number (N) and duration (D) of interruptions to the flow of traffic moved per unit of time. This problem can be viewed as

$$f(x_0) = \begin{cases} F(V_0, N) \\ F(V, N_0) \end{cases}$$

where $f(x_0)$ may represent a point or a solution set as is likely in this case. The solution set is described as that set in which improvements in one variable can only be gained at expense of the other variable. This is similar to Markowitz's efficient set.⁶ Also see Geoffrion⁷ and Charnes and Cooper⁸ on the analysis of non-solvable problems, "goal-attainment", as a safeguard to ensure that long-run or broader objectives are not obscured by immediately attainable objectives.

However, given the existing organization of the urban system, much of the systems analysis efforts with regard to traffic would take place outside of the police department. The possible technological spin-offs to the problem of traffic would be in the area of queuing and simulation models, the advent of the third generation series of computers now makes

many of these models economically feasible; a means of varying the length and sequencing of traffic signals in response to changes in the volume of traffic, through either a central control system or a device which responds to local conditions, and perhaps a simple information gathering system wherein the police officer on patrol reported possible future problems. Some examples might be trees or bushes growing in front of stop signs, or increased traffic due to a new factory or office. This information could form the inputs for work scheduling programs for both engineering studies and road maintenance.

Apprehension

It is in an evaluation of the effects of improving our ability to apprehend criminals that the broad outlook of the "true" systems approach is most revealing. It is assumed that the objectives of a police department ought to be based upon maximizing the safety of the individual and his property from harm by other individuals through either criminal or non-criminal means while attempting to minimize the quantity of interference with the public as it goes about its daily routine. Quantity in this case should be defined in terms of a combination of the number and duration of the interferences. The existence of multiple criteria implies a mapping of the criteria onto some function which *represents* the respective urban society's values. The previous comment regarding multiple goal-attainment again applies.

Although it appears that computer technology may be best suited for implementation in this function, without a careful analysis and evaluation of all the effects, many of the problems encountered in implementing computers into the production and inventory control functions in business may be repeated. And the results from "obvious" computer applications may cause a considerable delay in the development of this market.

Apprehension serves two purposes—to recover property, and to deter criminal activity through a high probability that the criminal will have to make some contribution to society in return for committing the crime. If, in fact, deterrence is the primary objective of apprehension, it is necessary that the detective division prepare the case; the judge, given the legal constraints set by the legislature or other appropriate set of elected officials, pass a long enough sentence to serve as a deterrent; prison

officials provide an internship which reinforces the deterrent effect without producing hardened criminals;⁴ and the probation officials' supervision of parolees be such that they, the parolees, have both the legitimate alternative activities to pursue, and the motivation not to return to criminal activities.

With the exception of murder and other crimes of passion, the criminal seems to be fairly successful in avoiding apprehension and subsequent conviction (see Table II).

The other problems of increasing our ability to apprehend are of two varieties. One, when those activities that are engaged in by the professional criminal are made less desirable, the criminal typically searches for new forms of criminal activity. Two, the cost to society of removing an individual is exceptionally high. This cost is composed of the following:

1. The cost of apprehension (this activity has the highest cost per assignment of all police department activities).⁹
2. Cost of preparation of cases, court appearances of both civil servants and citizens, and the maintenance and operation of the judicial system.
3. Cost of incarceration.¹⁰
4. Loss of income to the economy.
5. Possibility of family becoming a burden to the state.

Table II.

Crimes against the person		Cleared
Not Cleared		
9.8%	Murder	90.2%
14.5%	Negligent Manslaughter	85.5%
25.7%	Aggravated Assault	74.3%
33.1%	Forcible Rape	66.9%
Crimes against property		
63.0%	Robbery	37.0%
74.9%	Burglary	25.1%
73.7%	Auto Theft	26.3%
80.6%	Larceny	19.4%

Crimes Cleared by Arrest, 1964. Although, in actual numbers of occurrence, crimes against the person are infrequent, a high proportion of them lead to arrest. In cases of assault and rape, the victim may be able to identify the offender: murder and manslaughter, usually unplanned in advance, may have been witnessed by others. Also, the police exert great effort to solve these crimes. Based on Federal Bureau of Investigation, *Uniform Crime Reports for the United States*, 1964.

Naturally the cost to society of the criminal activity must be balanced against the above costs. I think the implicit hypothesis is that improved methods of apprehension would lead to a disproportionate number of convictions of the non-professional who commits the lesser crimes against society, and an even greater degree of control by government over individuals engaging in private, non-criminal pursuits. A further implication is that the professional criminal becomes rapidly aware of the improvements in the methods of detection and takes appropriate countermeasures or undertakes criminal activities in new forms of crime or in new geographical areas. If we are able to find improvements in the methods of apprehension that do not contribute greatly to the above, then it would seem that this would be a justifiable area of study.

Surveillance (Patrol)

The officer on patrol is a unique organizational entity. He can be considered to be a manager, if one defines managers as individuals whose primary function is decision-making, and yet, he is the lowest level operational member of the police department. He is required in any given day to make a large number of immediate decisions on what are, in fact, exceedingly complex questions. He has very little recourse to outside assistance to evaluate the consequences of his decisions, and he must make these decisions on the basis of highly uncertain and incomplete information and assertions. Bristow¹¹ notes at least four criteria on which the patrolman decision-maker must evaluate his alternatives: (1) legal, (2) precedent, both judicial and public policy, (3) press and public relations, and (4) internal (police department) relations.

It would appear, if we assume that it is possible to assign officers to patrol activities which tend to accomplish the derivative objectives of the public safety objective, that one way to improve the efficiency of the patrolman, decision-maker in the field, is to increase his ability to rapidly determine the consequences of the various alternatives in relation to any given decision or decision process.

To effectively provide the officer on patrol with information concerning consequences, we need both an information system capable of storing and receiving a large variety of information rapidly and making series of calculations for the field decision-

maker, and an efficient means by which the patrol officer can communicate with the system.

The NASA and DOD developed computer systems for automatic checkout and spaceflight control systems should provide the means of acquisition, reduction, and analysis of the inputs for the former, and for the latter, the devices which the astronauts use to both transmit and receive information as well as those sensing devices which automatically report changes in the ship's external and internal environment would seem to be especially applicable.

A typical inquiry made by an officer of the system might be to request the precedents, given some set of conditions of arrest and/or search of a suspicious individual when the officer did not have a search warrant and the effects upon later acceptability of the evidence. If the system indicated that it would not be advisable to enter without a warrant, it could indicate the most efficient means to obtain same and an estimate of how long it would take to obtain a warrant. As a longer range proposal, given appropriate revision of existing law, it might be possible to transmit the circumstances to a judge and to receive a facsimile warrant through equipment in the patrol vehicle.

Prevention

"Certain offenses may be considered as 'Preventable' while others may not. Crimes such as homicide, rape, aggravated assault, and others because of the passions (as opposed to reasons) usually motivating them, are not responsive to the mere knowledge that the police are present and will take steps to prevent them. Offenses in the general group containing crimes of stealth such as burglary, theft, auto theft, etc., on the other hand, do seem to respond to increased preventive effort."* (See Table II).

It is the very nature of crimes of stealth which makes efforts of apprehension relatively ineffective. The criminal who operates in this area carefully selects those places, situations, time and types of property in which the probability of apprehension is at a minimum. It would seem to be particularly relevant for a systems analysis effort to focus on the conditions under which this type of crime is most likely to occur.

* Quoted from R. Smith—1961, pp. 35-36. Also, Smith cites a study which indicates that "... special indirect techniques seem to have a fair amount of success in reducing the frequency of certain crimes related to vice." from *Narcotics, Addiction and Nalline*, Oakland, Calif. Police Dept., 1958, p. 10, (R. Smith, p. 36).

The recommendations as to the means of preventing crimes of stealth could take a number of forms:

1. Provide additional police protection in areas with a high likelihood of crime. This may very well require a resource allocation system with an exceedingly short reaction time and an accompanying system capable of rapidly recognizing changes in aggregate crime patterns.
2. Changes in the penalties for conviction of crimes of stealth. It has been in this area of prevention that much of the effort of law enforcement officials has already been applied.
3. Reduce the ability of criminals to utilize or dispose of stolen merchandise.
4. Statistical studies to indicate the causal conditions in which this group of crimes occur. The effect of street lighting and other physical attributes on crime levels is presently known. Cost effectiveness analyses would indicate the value of additional lighting particularly in areas and locations having an abnormally high incidence of crime. The use of direct warning systems in armored trucks, liquor stores, service stations and supermarkets would be a considerable deterrent from the substantially increased probability of apprehension.

All of the above with the exception of the changes in penalties for conviction would appear to have a good probability of success and significant applications of technological spin-off.

The suggestion for a resource allocation system could be accomplished through systems similar to the Defense Department-developed command and control systems. The system could use computer graphics to display the existing deployment of forces, both professional and equipment; geographical areas with increased intensity of crimes; and the number and present assignment of officers who could be relocated to help combat crime in the areas of increased intensity.

The system could also present a periodic (hourly, daily, weekly, etc.) allocation of departmental resources by area; e.g., beat. This might be used to

re-allocate officers to new beats, other divisions, etc., in the face of changing requirements. The periodic requirements information (based upon variations in the level of crime, as an example) could be fed into the system and an alternate resource allocation could be proposed which would reduce the "average gap between resources and requirements for each beat on division." Such an allocation could indicate the sources and destinations of the resources to be transferred, so that the administrative and supervisory units could determine the feasibility of the allocation. The process could conceivably be an iterative process wherein the supervisory officers constrained the system from allocations which they felt were not practical.

This system critically depends upon a means of measuring "normal" levels of activity, wherein the normal is defined in dynamic rather than static terms.

The design of systems with similar capabilities has been accomplished by DOD in its SAGE system and its tactical command and control systems, and NASA in its space flight control systems so that the problem would appear to be one of implementing existing technology into a civilian use.

The suggestion to reduce the ability of criminals to dispose of stolen goods could be accomplished through at least two means:

1. By marking easily disposable items with the owner's name and city. The methods of marking part numbers on components of space vehicles should provide the technology to develop inexpensive and innocuous means of identifying personal property either at point of sale or at police department approved locations.

2. By requiring better identification of sellers of expensive goods and requiring transfer of identifying information or original seller to future bills of sale. The knowledge, by criminals and "fences", that this information would be periodically processed and analyzed to check on multiple sales by individuals not normally engaged in buying and selling these items, and types of purchases made by pawnbrokers, jewelers, and other "legitimate" sources, might serve as a significant deterrent to this type of crime. In the opinion of many operatives and authorities in the field, it is the relative inability of the police to analyze and trace through economically, the massive amount of data on these items which makes apprehension so unlikely.

The direct warning system could be required by law of establishments under existing licensing requirements, in much the way owners of attractive nuisances are required to take precautions. The means to communicate the basic information to field (patrol car) could be accomplished through a localized real-time application of the suggested command and control system. Also, variations of missile automatic check-out systems could be used to perform many of the existing routine patrol activities, thereby releasing patrol officers to perform other functions. Another type of warning system which could directly use space effort developed technology would be a burglar alarm with a self-contained power source which could be charged daily during working hours. The device could be much like the meteor warning devices or a form of moving object radar similar to that used in aircraft ground control operations. Rather than transmit the impulse of movement to a cathode ray tube, as is the case of radar, a radio or telephone transmission would be initiated.

It would seem from the alternatives in this rather preliminary study, there would be a large number of possible applications of space and defense technology in the area of crime prevention.

An Example of a Possible Systems Solution to a Problem in Public Protection

Professional auto theft, in Southern California at least, in response to increased efforts by law enforcement officials is taking new forms. Typically an auto is stolen, stripped of most of its valuable and salable items and an attempt at destruction is made. Those items which can easily find their way into legitimate outlets are, of course, the most desirable. (The need for registration certificates for the sale of complete autos and engines makes this type of theft less desirable with the existing means available to check the certificates' validity.) At present the "hottest" item is a set of bucket seats from top-line Chevrolets, Pontiacs and Fords. A front and back set of these seats sells new for approximately \$800.00 with an eight to ten week wait for delivery from the factory. These items have now reached a "used" price from salvage dealers of about \$500.00 per set. A survey of the recent issues of the Los Angeles and Orange Counties Auto Wreckers News indicated that there were approximately three times as many seats of this type for sale as autos of same

make and year. When it is considered that not all of the models sold are equipped with the luxury seats, the ratio approaches ten to one—rather convincing evidence that stolen goods are getting on the "legitimate" market.

At present, the licensed auto wreckers are required to obtain a bill of sale for the items, but they are not required to check the identity of the seller. Since the seats may go through two to three auto wreckers before reaching the wreckers who specialize in this type of equipment, it becomes impossible to trace the original source.

After some consideration and further discussion, the following possible solution was presented to several knowledgeable law enforcement officials:

1. Require that on all automobile parts sales by a private party of over \$100.00 that the seller present a driver's license or similar identification at time of sale and that this be listed on the bill of sale and any further transfer of bill of sale.
2. If there exist any serial numbers that they be listed on the bill of sale.
3. The license (registration) of the vehicle from which equipment was removed be shown on the bill of sale. Seller must present proof of ownership (i.e., presentation of registration certificate). At this time the buyer would be required to ascertain face validity by comparing type of equipment to description of auto on the registration certificate.
4. Buyer of equipment who ultimately installed equipment would forward copy of bill of sale to Department of Motor Vehicles.
5. Periodically tests could be run using batch processing of the relevant information to determine:
 - (a) If the equipment could have come from auto (run against registration file).
 - (b) If the same equipment came out of a given auto more than once.
 - (c) Comparison of auto demolition records and equipment sales.
 - (d) Analysis of original "legitimate" buyers.

Most officials felt, based upon experience with pawnbrokers, that the above methods would probably have a considerable deterrent effect if they could be implemented!

The Problems of Implementing Information Systems

The usual approach of requesting users to report their information requirements to the analyst is primarily aimed at reducing duplication of efforts in collecting and reporting information. This approach has a tendency to neglect evaluating the value of the information to organization in terms of its contribution to the accomplishment of organizational objectives. If the studies made by the planning and research groups of municipal police agencies are an indication,⁴ any information system developed with the above approach will relate primarily to apprehension, and will be collected to facilitate administrative and supervisory needs rather than operational requirements (i.e., field decision making).¹²

Until the analysts are able to adequately understand the real problems in trying to assure public safety, and are able to suggest more effective alternative means which are focused on maximizing the safety of the inhabitants of the urban environment, any information system would most probably rely on the user requirements approach. However, the steps necessary for the implementation of the tactical control system suggested in the previous section on crime prevention would allow the team to view the system from a resources and requirements outlook. By approaching the design of an information system as a derivative function of a tactical control system, it would be practically assured that the informational requirements of the operational units would receive primary consideration. The ability of electronic data processing equipment to rapidly compile large quantities of data should be adequate

to meet administrative, supervisory and statutory requirements for information.

REFERENCES

1. Renis Likert, *New Patterns of Management*, McGraw-Hill, New York, 1961.
2. R. A. Johnson, F. A. Kast, and J. E. Rosenzweig, *The Theory and Management of Systems*, McGraw-Hill, New York, 1963, p. 350.
3. O. W. Wilson, *Police Administration*, 2nd Ed., McGraw-Hill, New York, 1963.
4. V. A. Leonard, *Police Organization and Management*, 2nd Ed., Foundation Press, Brooklyn, N.Y., 1964, p. 459.
5. W. W. Herrmann et al, "Natural Language Computer Processing of Los Angeles Police Department Crime Information," TM-17931000100 Sys. Development Corp., Santa Monica, Calif., 1 April 1964, p. 66.
6. H. Markowitz, *Portfolio Selection*, Wiley, New York, 1959.
7. A. Geoffrion, "On Solving Bi-Criterion Mathematical Programs," Working Paper 92, Western Management and Science Inst., UCLA, Los Angeles, 1965.
8. A. Charnes and W. W. Cooper, *Management Models and Industrial Application of Linear Programming*, Wiley, New York, 1961, pp. 215-233.
9. R. D. Smith, "Computer Application in Police Manpower Distribution," Washington Field Service Div., International Assoc. of Chiefs of Police, 1961, p. 98.
10. L. Glen Strasburg, Personal discussion with regard to his professional work with the Calif. State Dept. of Corrections, January 1966.
11. Allen P. Bristow, "A Preliminary Study of Problems and Techniques in Decision-Making for the Police Administration," School of Public Administration, U.S.C., Los Angeles, 1957.
12. Oscar Speed, "A Report of an Administrative Analysis of a Police Car Reporting System," School of Public Admin., U.S.C., Los Angeles, 1961.

RECENT PROGRESS ON A HIGH-RESOLUTION, MESHLESS, DIRECT-VIEW STORAGE TUBE *

Norman H. Lehrer and Richard D. Ketchpel

*Hughes Research Laboratories,
Malibu, California*

INTRODUCTION

Photographic film is an excellent display medium, with a wide dynamic range, high resolution, and an integration capability. It has some limitations, however, because it is not a real-time medium, i.e., the signal is not visible immediately upon application to the film; at least several seconds must elapse before the recorded signal is developed. In addition, film is not reusable; consequently, when no permanent record is required, the use of film is extremely wasteful.

Development of a device with display capabilities which might be described as "real-time reusable photographic film" has long been a goal of display research. Several approaches have been investigated unsuccessfully.

PREVIOUS APPROACHES TO REAL-TIME, HIGH-RESOLUTION CONTROLLED STORAGE AND DISPLAY OF INFORMATION

The simplest of these approaches is the long persistence phosphor screen, such as the P7 or P14. These screens achieve storage by the cascading of

* The research reported here was partially supported by the Research and Technology Division, Air Force Systems Command, U.S. Air Force.

phosphors, so that the cathodoluminescence of a short decay blue-emitting phosphor excites the photoluminescence of a long decay yellow- or orange-emitting phosphor. The storage characteristic of a P7 phosphor is indicated in Fig. 1. Note that the

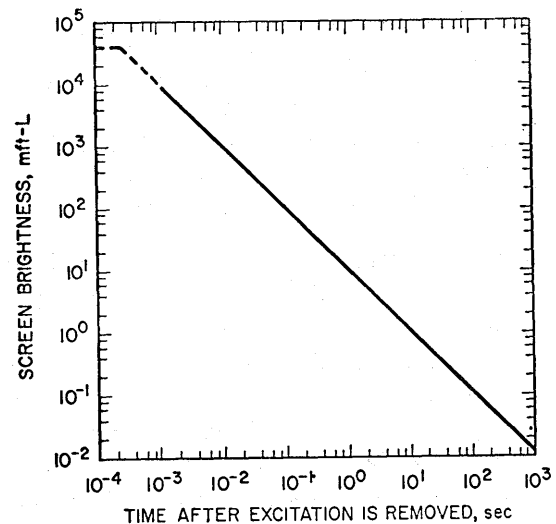


Figure 1. Decay of a long persistence phosphor.

initial decay of these screens is so rapid that the brightness is on the order of millifoot-Lamberts only 1 second after excitation ceases, requiring dark adaptation for viewing. The resolution is limited by the spreading of the emission which excites the sec-

ond phosphor, and the persistence is not controllable because it is a characteristic of the material. The limitation of the storage capability of these long-persistence screens can be understood quite simply when we consider the energy available for excitation of the phosphor. The only source of energy is that imparted to the phosphor during excitation by the signal delivered with an electron beam. To achieve storage, this relatively large amount of energy delivered in a short time must be reemitted for much longer periods. Even in the ideal case, this can occur only if the stored brightness level is substantially less than that achieved during excitation.

In direct-view storage tubes an attempt is made to overcome these fundamental limitations on brightness by utilizing a second source of energy to maintain the display. The energy delivered by the signal is used only to produce a modulation on the second source of energy. In such a device, as shown in Fig. 2, a second energy source is provided by separate

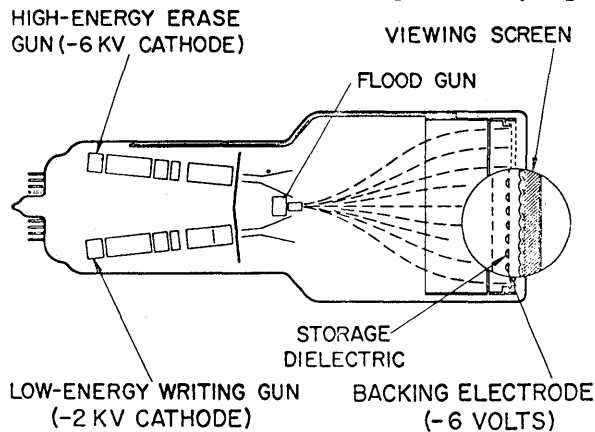


Figure 2. Schematic of storage grid tube.

rating the persistence characteristic from the brightness characteristic of the display; i.e., utilizing the phosphor only for its brightness. Storage is achieved through the addition of a storage mesh. A storage dielectric resides on the storage mesh, and signals are displayed and stored by the charge pattern which exists on the surface of the dielectric. A flooding beam which serves as a second source of energy is modulated by the charge pattern on the storage mesh, creating the stored image on the viewing screen. Therefore, in a direct-view storage tube it is possible to store information at high brightness levels and controllably erase the display. However, the nature of the structure which makes possible the controlled display of information at high brightness

levels in the direct-view storage tubes also limits the resolution. The geometry of the mesh and front end electron optics limits the resolution of a direct-view storage tube to about 120 lines per inch, or more than an order of magnitude below that obtainable with photographic techniques.

One approach to overcoming this resolution limitation is the electroluminescent-photoconductive panel, as shown in Fig. 3. In such a device, a layer

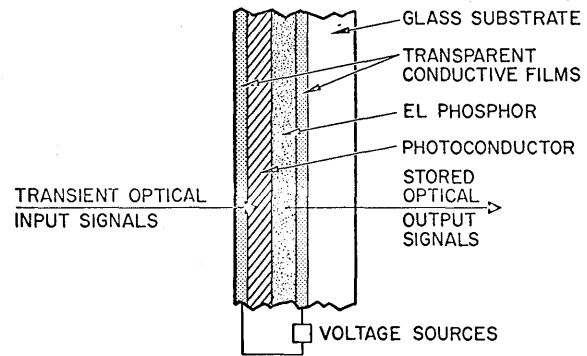


Figure 3. Principle of EL-PC storage by feedback.

of luminescent material is adjacent to a layer of PC material; these layers, in turn, are enclosed by two transparent electrodes. The pulsed input signal increases the conductivity of the photoconductor, switching more voltage across the EL layer and causing it to light. The EL layer remains lit after the input signal ceases because light from the EL layer feeds back to the photoconductor, maintaining its high conductivity. In practice, this concept is limited in response time and resolution. Conventional photoconductors respond in milliseconds, compared with the microsecond excitation required in many applications. In addition, the spreading of the light which feeds back from the EL layer to the photoconductor causes a severe degradation in resolution. The resolution problem might be solved by the use of a mosaic structure of EL-PC elements (Fig. 4), separate

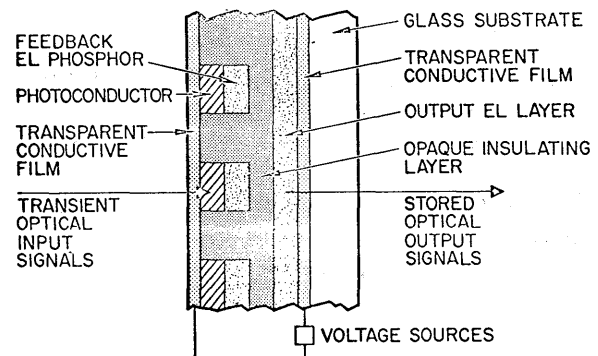


Figure 4. Modified EL-PC structure to prevent spreading.

rated from each other by an opaque insulating layer, and in series with an output EL phosphor. Such complex structures restrict the resolution below that obtainable with conventional direct-view storage tubes. Thus the essential drawbacks to the EL-PC panel may be summarized as a lack of sensitivity, because of the nature of photoconductivity, and a lack of resolution, because storage is accomplished by feedback of light to a photoconductor.

If the storage capability were an integral characteristic of the control layer, it would be possible to optically isolate that layer from the EL layer simply by inserting an opaque insulating layer between the control and EL layers. In such a case, the resolution would be ultimately determined by the inherent storage characteristics of the control layer, and it would not be degraded by the requirement for optical feedback. Furthermore, if the control layer could respond to microsecond electron beam excitation, it would be possible to fabricate a device which could store and display information in real time with high resolution and high sensitivity. Thin dielectric films which exhibit sustained electron bombardment induced conductivity (SEBIC) appear to satisfy the control layer requirements for high sensitivity and storage.

THE SEBIC LAYER

Thin films of cadmium sulfide which exhibit SEBIC were first developed by the Hughes Research Laboratories. The principal phenomenological characteristics¹ of these layers can be summarized as follows:

- Pronounced rectification effects are observed when an electric field is applied without external excitation (i.e., light or electron beam). Rectification ratios as high as 10^6 are obtained.
- When the direction of the applied field places the layer in the back biased or low current condition, external excitation produces a substantial increase in the conductance of the excited region. The amount of the increase depends on the integrated excitation energy.
- When the external excitation is removed, but the application of the field continues, the high conductivity state is substantially maintained in the previously excited region.

- Momentary reversal or removal of the field restores the excited region of the layer to its low conductance state.
- The conductance of the SEBIC layer can be varied in two dimensions for long periods with no significant loss in the stored resolution (which approaches that available with photographic techniques).
- The conductance of the SEBIC layers can be substantially increased by bombardment with microsecond pulses of high energy electron beams.

A cross section of a typical SEBIC layer is shown in Fig. 5. The SEBIC layer consists of a layer of

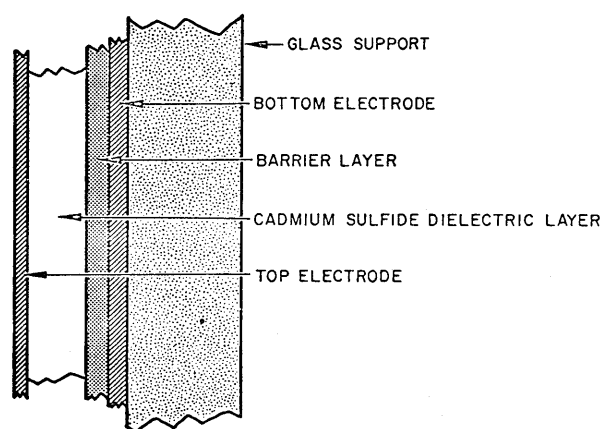


Figure 5. Cross section of SEBIC layer.

evaporated cadmium sulfide sandwiched between two electrodes. A barrier region is formed adjacent to one of the electrodes. The electrical characteristics of the SEBIC layer critically depend upon the formation of the barrier region. In the SEBIC layer shown in cross section in Fig. 5, this barrier region is formed adjacent to the bottom electrode, but it can also be made adjacent to the top electrode.

Theoretical concepts of the SEBIC effect are described elsewhere;¹ however, the effect is not yet understood completely.

SEBIC layers can store information in the form of two dimensional conductivity modulations with almost photographic resolution. In addition, they can be excited with brief pulses of high energy electron beams, and they are reusable because they can be erased almost instantaneously. In a sense, they may be thought of as a form of real time photographic film—the principal remaining problem concerns the readout of information. One possible tech-

nique involves utilizing the conductivity modulations stored in the SEBIC layer to switch on an adjacent EL layer, or, in other words, replacing the PC layer in an EL-PC panel with a SEBIC layer. The successful realization of this concept depends upon appropriately matching the impedance of the EL and SEBIC layers. This approach—the EL-SEBIC panel, or the meshless storage screen—is described below.

THE MESHLESS STORAGE TUBE

Description

The meshless image storage screen is based on the use of the phenomenon of SEBIC to control the brightness of an adjacent EL phosphor. As shown in Fig. 6, the two layers are sandwiched be-

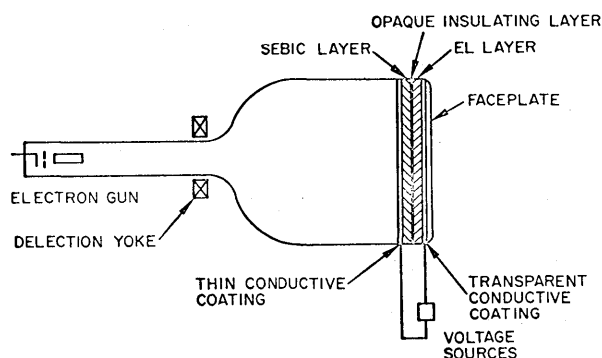


Figure 6. Schematic of meshless storage screen.

tween two electrodes. A transparent conductive film deposited on a glass support serves as the electrode contacting the phosphor; a thin conductive coating deposited on the SEBIC layer forms the other electrode. In contrast to the construction of conventional PC-EL phosphor panels, optical feedback is prevented by the use of an opaque insulating layer between the SEBIC and phosphor layers. The opaque layer can be omitted if the SEBIC layer is insensitive to the luminous output of the phosphor layer.

The switching action is generally similar to that of PC-EL phosphor panels. When voltage is applied to the electrodes, most of the voltage drop is across the SEBIC layer when it is unexcited; the EL layer is dark if the impedances of the two layers are properly selected. When an elemental region of the SEBIC layer is excited by means of a high energy electron beam, its resistance decreases; more of the voltage drop is switched across the EL layer, caus-

ing it to light. Because the conductivity of the SEBIC layer remains high after excitation ceases, the EL layer remains lit and the signal is stored. The display can be erased instantaneously by a momentary removal of the applied voltage, which permits the SEBIC layer to return to its pre-excited high resistivity value. The form of the decay can be controlled by low duty cycle interruptions in the applied voltage.

Analysis

The equivalent circuit of an elemental region of the meshless storage screen is shown in Fig. 7. The

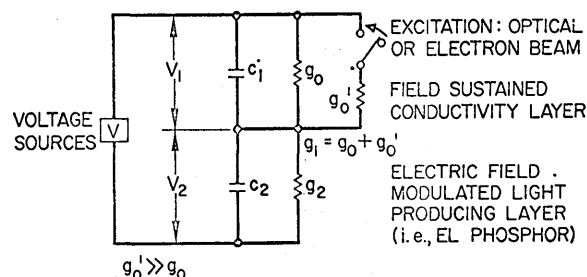


Figure 7. Equivalent circuit of meshless storage screen.

SEBIC and EL layers are represented by their capacitive and conductive components. The SEBIC layer has an elemental capacitance c_1 , an unexcited conductance g_0 , and an excited conductance g'_0 which is much greater than its unexcited conductance; the total elemental conductance is g_1 . Note that the effect of electron beam excitation is represented by closing the switch in series with conductance g'_0 . Because of the nature of the sustained conductivity in the SEBIC layer, the switch remains closed after excitation ceases, as long as the applied field is maintained. The switch is opened by momentary removal of the applied field. The elemental capacitance and conductance of the EL layer are represented by c_2 and g_2 , respectively.

The voltage sources (shown in Fig. 7) for driving the meshless storage screen are a dc bias in series with an ac voltage supply. The dc serves to prevent the ac signal (which excites the EL phosphor) from reversing the field across the SEBIC layer and thereby erasing any stored signals.

Control by the SEBIC layer of the voltage drop across the EL layer implies that most of the voltage drop initially is across the SEBIC layer; as a result of excitation, a good part of the drop is switched across the EL layer. The brightness versus voltage characteristic of the EL layer will determine how

much must be switched; a reasonable assumption is that when the EL layer is dark, two-thirds of the voltage drop should be across the SEBIC layer and when two-thirds of the drop is across the EL layer, it is at highlight or full signal brightness. The problem lies in determining the optimum impedance relationships which will permit the required voltage switching across the EL layer. The ac switching problem will be treated first, and then the dc problem.

If V_1 and V_2 are the ac peak-to-peak voltage drops across the elemental regions of the SEBIC and EL layers, ϕ is the ratio of their absolute values;

$$\phi = \frac{|V_1|}{|V_2|} \quad (1)$$

ϕ can also be expressed as

$$\phi = \frac{|V_1|}{|V_2|} = \frac{|z_1|}{|z_2|} = \frac{|y_2|}{|y_1|} \quad (2)$$

where z_1 , z_2 and y_1 , y_2 are, respectively, the impedances and admittances of the two layers.

The elemental admittance y_1 of the SEBIC layer is

$$y_1 = g_1 + \omega c_1 j, \quad (3)$$

and the elemental admittance y_2 of the EL layer is

$$y_2 = g_2 + \omega c_2 j \quad (4)$$

where ω is the frequency of the applied ac signal multiplied by 2π . Therefore,

$$\phi = \frac{|V_1|}{|V_2|} = \frac{|y_2|}{|y_1|} = \frac{\sqrt{g_2^2 + \omega^2 c_2^2}}{\sqrt{g_1^2 + \omega^2 c_1^2}}; \quad (5)$$

squaring both sides of (5),

$$\phi^2 = \frac{g_2^2 + \omega^2 c_2^2}{g_1^2 + \omega^2 c_1^2}. \quad (6)$$

From (6), the conductance of the EL layer g_2 is typically small compared with its susceptance ωc_2 . Furthermore, in the off condition, the conductance of the SEBIC layer g_1 is small compared with its susceptance ωc_1 . In that case, therefore, the voltage division is inversely proportional to the ratio of the elemental capacitance of the two layers:

$$\phi \text{ (off)} = \frac{c_2}{c_1}. \quad (7)$$

Therefore, if most of the voltage drop is to be across the SEBIC layer in the off condition, the capacitance of the EL layer must be large compared with that of

the SEBIC layer. SEBIC layers are typically 5 to 10 μ thick, resulting in a requirement for EL layers which are 2½ to 5 μ thick; based on the state of the art, ac EL layers with this thickness do not appear practical for application to a device. In addition, the SEBIC layer does not store well under ac operation because of the periodic reductions in the field produced by the ac voltage.

The operation of the meshless storage tube can be analyzed for the dc case by assuming the voltage source in Fig. 7 is dc. The immediate voltage division upon application of the dc bias V is determined by the inverse ratio of the elemental capacitances of the two layers; the steady state voltage division is determined by the values of the resistances. The time constant of the circuit must be measured in milliseconds (or less) to permit rapid erasure of the display. The voltage across the EL layer V_2 is given by

$$V_2 = \frac{r_2 V}{r_2 + r_1} - \left(\frac{r_2}{r_2 + r_1} - \frac{c_1}{c_1 + c_2} \right) e^{-t/\tau} \quad (8)$$

where the time constant

$$\tau = \frac{c_1 + c_2}{1/r_1 + 1/r_2} = \frac{r_1 r_2 (c_1 + c_2)}{r_1 + r_2} \quad (9)$$

and

$$g_1 = \frac{1}{r_1}, \quad g_2 = \frac{1}{r_2}.$$

It is reasonable to assume that to calculate the order of magnitude of the time constant τ , the capacitances of the SEBIC and EL layers are approximately equal since their thicknesses and dielectric constants are comparable. Therefore,

$$c_1 \cong c_2 \quad (10)$$

Similarly, in the erased condition, the resistivity of the SEBIC layer is of the same order of magnitude as that of the EL layer:

$$r_1 \cong r_2. \quad (11)$$

Substituting these values in the equation for the time constant,

$$\tau = \frac{r_2^2 2c_2}{2r_2} = r_2 c_2 \quad (12)$$

The resistance of the EL layer can be expressed as

$$r_2 = \frac{d}{A} \rho_2 \quad (13)$$

where d is the thickness of the layer, A is the area of

the element, and ρ_2 is its resistivity. The capacitance of the EL layer can be expressed as

$$C_2 = \epsilon \epsilon_0 \frac{A}{d} \quad (14)$$

where ϵ is the dielectric constant and ϵ_0 is the permittivity of free space.

Substituting (13) and (14) in (12),

$$\tau = r_2 C_2 = \frac{d}{A} \rho_2 \cdot \epsilon \epsilon_0 \frac{A}{d} = \epsilon \epsilon_0 \rho_2 \quad (15)$$

A reasonable value for ρ_2 is $10^6 \Omega\text{-m}$, and for ϵ the value is 10;

$$\begin{aligned} \tau &= 10 \times 8.85 \times 10^{-12} \times 10^6 \\ \tau &= 88.5 \mu\text{sec} \end{aligned} \quad (16)$$

Thus, a response time of only 88 μsec is achieved when $r_2 = 10^6 \Omega\text{-m}$. r_2 can increase by two to three orders of magnitude before the erase time becomes excessive.

Control by the SEBIC layer of the voltage drop across the EL layer implies that most of the voltage drop initially is across the SEBIC layer; as a result of excitation, much of the drop is switched across the EL layer. The brightness versus voltage characteristic of the EL layer will determine how much voltage must be switched. A reasonable assumption is that when the EL layer is dark, two-thirds of the voltage drop should be across the SEBIC layer; when the two-thirds of the drop is across the EL layer, it is at highlight or full signal brightness. The problem is to determine the optimum impedance relationships which will permit the required voltage switching across the EL layer. If V_1 and V_2 are the dc voltage drops across the elemental regions of the SEBIC and EL layers, let ϕ be the ratio of their values:

$$\phi = \frac{V_1}{V_2} \quad (17)$$

ϕ in the steady state condition can also be expressed

$$\phi = \frac{V_1}{V_2} = \frac{r_1}{r_2} \quad (18)$$

In such a case, V_1 changes linearly with r_1 . Obviously, if two-thirds of the voltage is to be across the SEBIC layer, r_1 must be twice r_2 ; conversely, if the two-thirds voltage is to be across the EL layer, r_1 must be $\frac{1}{2} r_2$. These impedance relationships are indicated in Fig. 8.

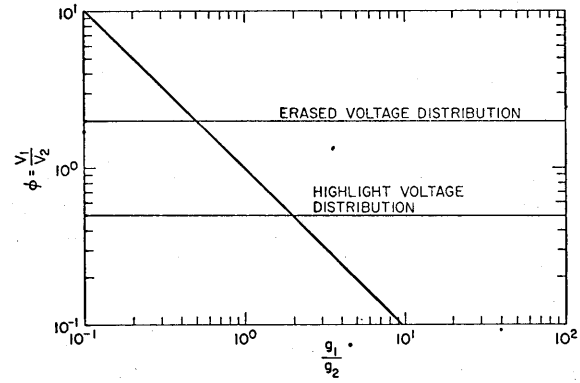


Figure 8. DC voltage control with the SEBIC layer.

SEBIC and EL Layer Characteristics

Substantial progress has been made in the fabrication of SEBIC and dc EL layers with matching impedance characteristics. The current-voltage characteristic of a high voltage SEBIC layer is indicated in Fig. 9; the area of the electrode was 22 mm.²

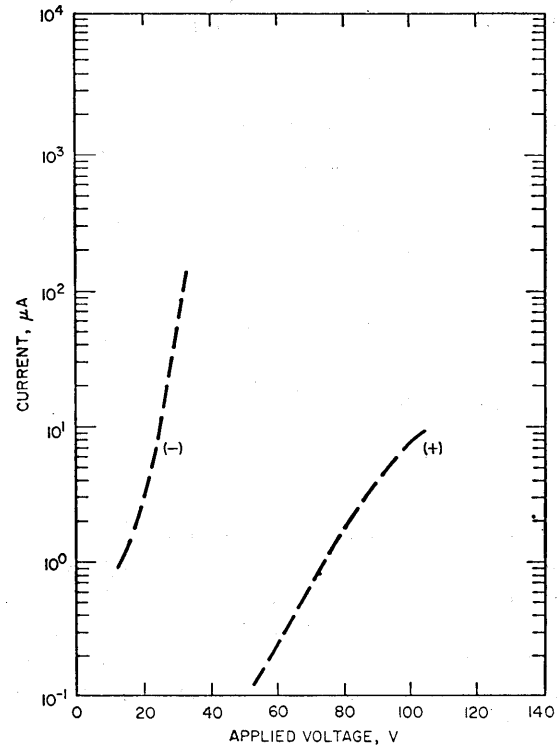


Figure 9. High voltage SEBIC layer current-voltage characteristic.

Note that operating voltages of 100 V and more have been achieved with stored current densities approaching 1 mA/cm².

DC electroluminescent layers are particularly promising for use with the SEBIC layer because of the dc nature of the storage characteristic. The thickness of the EL layer is no longer critical because the capacitance is of minor consideration in the impedance match. In addition, dc EL layers operate at much lower maximum voltages than those required for equivalent brightness from ac powder layers. For example, typical brightness for state-of-the-art EL layers is 10 ft-L at 20 V for dc layers and 100 V peak to peak for ac excited layers. The life of ac excited layers appears reasonable, however, while that of dc layers is highly questionable at present.

Several techniques for preparing evaporated dc EL films are described in the literature. One technique, described by Thornton,² utilizes an evaporated film of ZnS:Cu, Mn, Cl. Subsequently, the evaporated film is heat treated in a powder environment which also contains the appropriate activators. The second technique, described by Goldberg and Nickerson,^{3,4} involves the use of evaporated ZnS:Mn, Cl. In the following step, a thin copper layer is deposited. Finally, the composite layers are heat treated in vacuum to produce the activated film. Another technique described by Cusano involves preparation of the same ZnS:Mn, Cl films through the use of a vapor phase reaction process, followed by a copper diffusion step. In general, the technique of Goldberg and Nickerson was used as a starting point in our studies.

Operating characteristics for a dc EL layer are shown in Fig. 10. Voltage, current, and input power

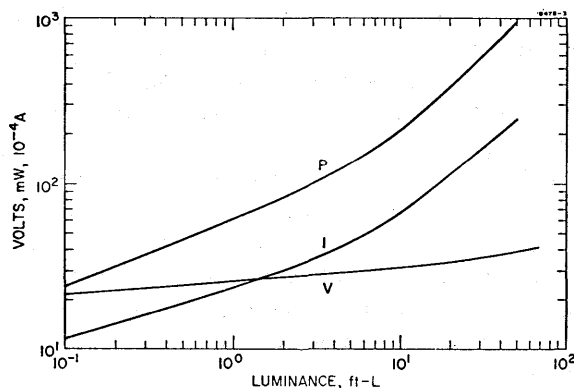


Figure 10. Luminance of a dc EL layer.

are plotted versus brightness on log-log coordinates. Note that the slope of the power versus brightness curve does not become equal to 1 until maximum light output is approached. Luminous efficiency is defined as brightness (ft-L) divided by input power (watts) per square foot of area, so that maximum

efficiency is reached as the slope of P versus B approaches 1.

Studies have been conducted which utilize SEBIC layers to control an equal area dc EL cell. In one such case, the potential applied to the series circuit of the dc EL layer and the SEBIC layer was 60 V. The induced brightness (or brightness during excitation) was 35 ft-L. The stored brightness 5 seconds after excitation ceased was 7 ft-L, and the erased brightness was 0.001 ft-L. This controlled storage of a dc EL layer for a 7000:1 brightness range obviously implies that a practical display can be produced.

APPLICATIONS

The applications of the meshless storage tube will ultimately depend on its performance. Based on the progress to date, it seems reasonable to assume that a meshless storage tube eventually will be fabricated which will store for several minutes with a resolution of up to 1000 lines/in. over screen diameters as large as 21 inches. Writing speeds of 1,000,000 in./sec appear feasible. Erasure can be instantaneous or slow over a controllable time period. The maximum light output might be 40 ft-L, with a high contrast screen permitting observation in high ambients.

Before examining the various applications, we should consider how much resolution the eye can utilize. It is frequently accepted that the eye can resolve elements about 1 mm apart. At a viewing distance of 10 inches, this is equivalent to about 200 to 300 lines/in. Therefore, a display device capable of producing an image with 300 lines/in. is required to match the resolution of the unaided eye. (This is more than twice the resolution which is realizable in the best mesh-structured direct-view storage tubes.)

With the aid of simple optical devices, magnifications of up to five times are practical. Therefore, a display resolution of 1000 to 1500 lines could be seen with such devices. (It should be noted that very high resolution systems frequently will have long frame times, permitting adequate time for close inspection with optical devices.)

The applications of the meshless storage tube can be arbitrarily divided into two broad areas, based on the nature of the information to be displayed.

Long Frame Time Systems

In these systems, the information becomes avail-

able and is serially delivered during frame times which exceed the storage time of the eye. Since the retention time of the eye is usually taken as 0.1 second, in these systems the rate is less than 10 frames/sec. Typical examples of such low frame rate systems are commercial and military ground-to-air radars. In these systems, the information is generated by the scanning radar and the frame time may range from several seconds to a minute. The resolution is usually more than 1000 lines. These radars normally utilize long persistence phosphor screens for the display. Therefore, the storage is at very low brightness levels and is not controllable. A substantial improvement in this display could be achieved by use of the meshless storage tube. The information would be written on the screen by a high energy electron beam, as in the case of an ordinary cathode ray tube. The information could be stored at a level close to its initial brightness over the entire frame, or could be made to decay in accordance with the frame time. This would result in a three order of magnitude increase in the stored brightness over that obtained with long persistence phosphor screens.

Bandwidth Compression Systems

These systems are characterized by the immediate availability of large amounts of information at a given location. The problem is to transmit this information to a remote location for display. If the transmission is over a phone line, the bandwidth limitation is approximately 4 kc; this means that the transmission of a television picture frame with 500 line resolution which normally occurs in a 1/30 sec takes place over 1/2 min. The conventional mesh structured storage tube is capable of this type of resolution. However, the meshless tube offers advantages in simplicity of operation and (potentially) substantially lower cost.

The transmission of the typewritten page or high quality pictures requires display resolutions of 1000 lines or more. The unique resolution capabilities of the meshless storage tube permit applications which are impossible with conventional storage tubes. The controlled persistence feature would permit extended viewing and erasure at will.

Narrow band transmission lines between large centrally located computers and many individual remote console displays should also be an important application for this device. The main features of this application would be the high resolution, flicker-free

picture without the requirement for a local memory at each console to regenerate a cathode ray television type of display. A specific use would be in a "lookup" type of operation where the operator is perusing typewritten or graphical data in a search for a particular item. The high resolution would allow a large amount of information to be presented on one format, so that the operator could scan the contents rapidly and select the portion in which he was interested. A large library or inventory might thus be stored in a central computer for examination at remote locations. It should be stressed that the high resolution capability of the device would permit the display of pictorial and graphic information.

An application which appears particularly promising is that which is frequently called "phonovision." The most important obstacle to the use of phone lines for a display of information with each phone has been the unavailability of a low cost, high resolution storage and display device which must be used with phone lines. The potential simplicity and low cost of the meshless storage tube indicate that such a system should be practical and could well be the largest application of the device.

CONCLUSIONS

Based on the use of a thin film which exhibits SEBIC to control the voltage drop across an adjacent dc EL phosphor, a high resolution meshless storage tube appears promising. Recent progress indicates that it should be possible to electrically control the brightness of the dc EL layer over a wide brightness range with a maximum stored brightness of 40 ft-L. Resolutions approaching 2000 lines/in. may be possible.

Other approaches to the problem of the high resolution storage and display of information have included cathode ray tubes with long persistence display screens, meshless-structure storage screens, and EL-PC panels. These approaches suffer from limitations in resolution, response time, or control of persistence.

The unique display characteristics, as well as the simplicity and potential low cost, of the meshless storage tube may lead to a wide variety of important applications. These include systems where the information is available in frame times which are long compared with the storage time of the eye (such as ground based radars). In another class of applications, information may be displayed at a location

remote from the source of the information (bandwidth compression). The high resolution capability of the display would permit a large amount of information to be stored in a computer and then transmitted to a remote location for display. This information could be pictorial, graphic, or alphanumeric. The low cost and simplicity of the device could make phonovision a reality.

REFERENCES

1. N. H. Lehrer and R. D. Ketchpel, "Thin Film Conductive Memory Effects Applicable to Electron Devices," *Optical and Electro-Optical Information Processing*, MIT Press, Cambridge, Mass., 1965, pp. 419 to 434.
2. W. A. Thornton, *J. Appl. Phys.* vol. 33, p. 1602 (1963).
3. P. Goldberg and J. W. Nickerson, *J. Appl. Phys.* vol. 34, (1963).
4. J. Nickerson and P. Goldberg, *Tenth National Vacuum Transactions*, Macmillan, New York, 1963, pp. 475 to 479.

ACKNOWLEDGMENT

The authors wish to express their appreciation to the Air Force Avionics Laboratory for their interest and sponsorship of this effort. Mr. William H. Nelson of that Laboratory deserves special thanks for his continued suggestions and encouragement.

THE PLASMA DISPLAY PANEL—A DIGITALLY ADDRESSABLE DISPLAY WITH INHERENT MEMORY *

D. L. Bitzer and H. G. Slottow

*Coordinated Science Laboratory
University of Illinois*

INTRODUCTION

Despite a growing interest in graphic communication in computer systems and a rapidly developing computer technology, the cathode ray tube remains the most useful of available display devices. Its limitations, however, are serious, particularly in systems with many display terminals. Other than phosphorescence it has no memory. Its images, therefore, must be regenerated continually, and to avoid flicker they must be transmitted at video bandwidths. Furthermore, as an analog device in a digital environment the cathode ray tube requires signal conversion circuits that are both complex and expensive. Other limitations such as high voltage and space requirements are less serious but are still significant.

The Plasma Display is a new device that, in contrast to the cathode ray tube, retains its own images and responds directly to the digital signals from the computer. Its resolution is comparable to that of cathode ray tube displays, and in addition it can be interrogated by the computer. It also seems likely

that images can be drawn directly on the display panel by means of a light emitting pencil. Although this device is at an early stage of development some of its system properties are already known, and others can be estimated. The purpose of this paper is to discuss, on the basis of these properties, the role that the Plasma Display could fill in future computer systems. A brief description of the display is also included. More detailed accounts are available,¹⁻³ and a discussion of the display from a device standpoint will be published elsewhere.

DESCRIPTION OF DISPLAY

The display is constructed of three pieces of flat glass as shown in Fig. 1. Through the center piece a matrix of small holes is drilled, and on one surface of each outer piece a grid of transparent gold conducting strips is vapor deposited. The glass sheets are typically 0.006 inch thick and the holes in the center piece are about 0.015 inches in diameter. In assembly the grids are on the outer surfaces of the panel, they are orthogonal to one another, and the strips are in registration with the holes. Each gas cell, shown in the section view of Figure 2, is thus surrounded entirely by glass except at the interfaces between the glass sheets. Through these thin gaps air is evacuated and a gas is admitted to the cells.

* Supported in part by the Joint Services Electronics Program under contract number DA 28 043 AMC 00073 (E), in part by the Advanced Research Projects Agency under contract number ONR Narr—3985 (08) and in part by the Syracuse University Research Corporation under contract number SURC 66124.

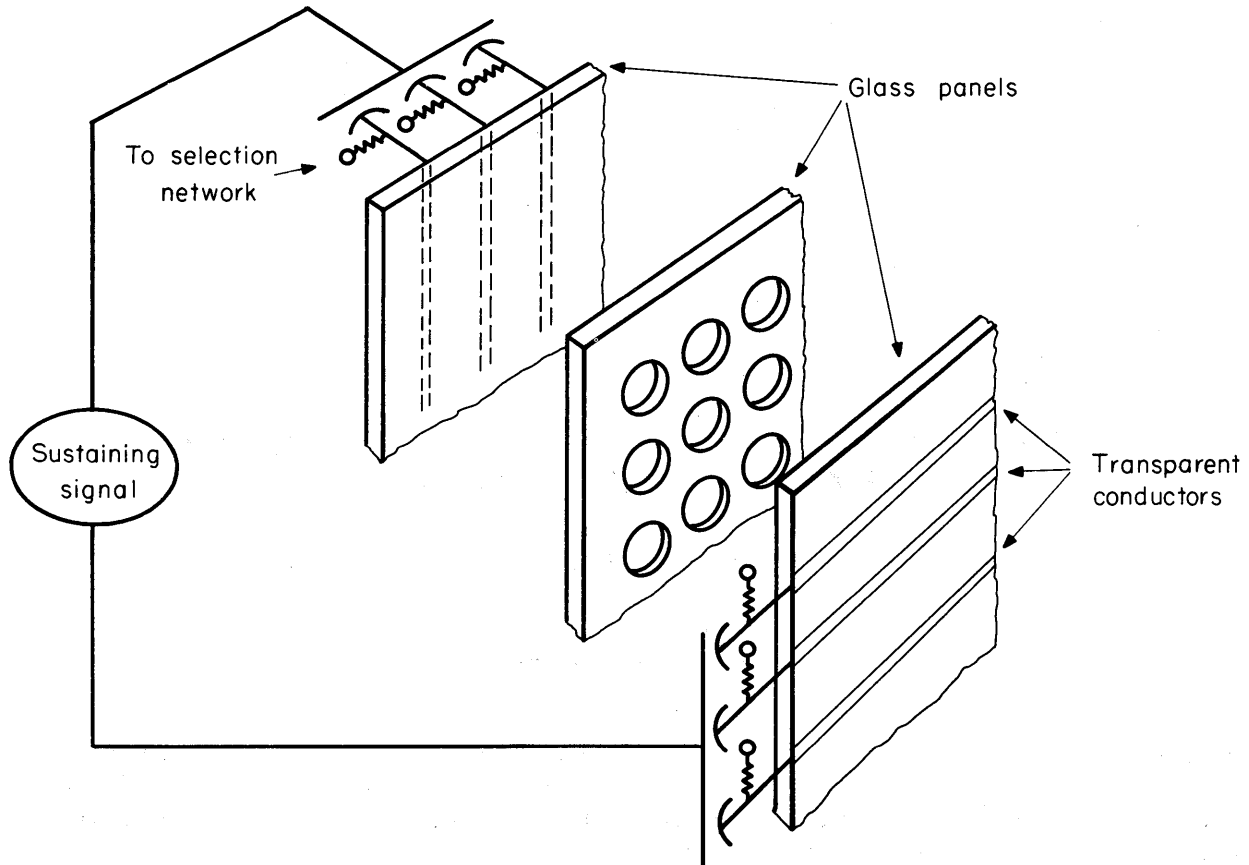


Figure 1. Assembly of plasma display panel.

If an alternating voltage across a cell exceeds the firing voltage, a discharge is established which develops rapidly to a glow. At the same time the flow of charges to the insulating walls reduces the voltage across the cell. When this voltage becomes too low to maintain the discharge, the glow is diminished and the discharge itself is quenched. Measurements of the current in the cell and of light radiated from the cell have shown that with appropriate gases this entire process takes place in from 50 to 75×10^{-9} seconds. During the following half cycle the voltage caused by the wall charge adds to the voltage due to the external signal. Once the first discharge has been established, therefore, the voltage required from the exciting signal to establish succeeding discharges is

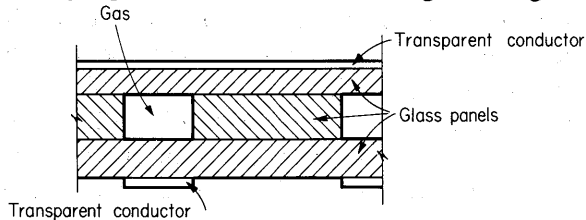


Figure 2. Gas discharge cell.

less than that required in the absence of wall charge. In fact, if the flow of charges to the walls is just sufficient to neutralize the field that exists within the cell at the time of firing, the ratio of these two voltages is 2:1. At intermediate voltages, therefore, the cell is bistable. In the "zero" or "off" state the peak cell voltage is insufficient to create a discharge. In the "one" or "on" state a brief discharge occurs once each half cycle of the sustaining signal. Figure 3 shows a photograph of the light pulses together with the exciting signal. The time scale is 1 microsecond/division, and the sweep is triggered repeatedly during the 1/25 second exposure time.

The memory actually resides in the wall charges,

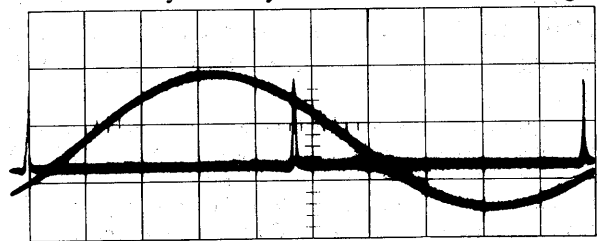


Figure 3. Light pulses and exciting signals.

and it is often convenient to describe the device processes in terms of the wall voltages associated with these charges. In the "off" state, for example, the wall voltage is, ideally, zero. In the "on" state the wall voltage alternates at the exciting frequency, combining once each half cycle with the external voltage to fire the cell. Changing the state of a cell is essentially a matter of controlling the changes in the wall voltage.

Except when the pattern on the display is changed a balanced alternating signal, either sinusoidal or pulsed, is always applied across the two grids. This sustaining voltage is in the bistable range, and the voltages on all lines in a grid are equal. During this time the pattern remains on the display and there is no communication between the computer and the display. When the state of a cell is changed an appropriate balanced voltage is applied across the two electrodes that intersect at that cell. Across the remaining cells affected by these electrodes only one half the voltage appears. This reduced voltage is within the bistable range and does not change the states of these cells. Write signals from the computer cause the states of selected cells to be changed in sequence from "off" to "on." Erase signals which change states from "on" to "off" control either single cells in sequence or rectangular blocks simultaneously.

If the peak voltage across two conductors that intersect at an "off" cell is raised above the firing potential the cell will be driven to the "on" state. In the process the wall voltage, starting at essentially zero, oscillates between two changing positive and negative values until after several cycles these limiting voltages reach stable values. The external voltage, however, need only exceed the firing voltage once to initiate the process.

The short transition time cannot be attributed to charge leakage along the glass surfaces. In fact charge can remain on the walls for many milliseconds. We have observed, however, that both the intensity of the discharge and the amount of wall charging increase when the slope of the exciting signal increases. The differential charging during the first few discharges after the state is changed drives the process to equilibrium.

The state of a cell can also be changed from "off" to "on" by combining a slowly varying control signal with the sustaining signal. This slow write procedure allows the use of high impedance switching cir-

cuits, and is appropriate to several important uses of the display.

Several procedures, both fast and slow, have been used to change the state of a cell from "on" to "off."

The computer not only can write and erase patterns on the display, but it can also test the state of any cell by applying the sustaining signal to only the electrodes that intersect at that cell. If the sustaining signal is applied to only one pair of intersecting conductors only that cell responds to the signal. If the cell is "on" it emits light pulses that are detected by a single photocell at the back of the display panel. If the cell is "off" it does not emit light. The states of all other cells remain unchanged since the wall voltages do not change appreciably in this time interval. If the sustaining signal is pulsed rather than sinusoidal the read signal is timed to appear between pulses. By sequentially testing selected cells the computer can read information from the display.

The initiation of a gas discharge requires both a sufficiently high voltage and the participation of charged particles. In normal operation electrons are provided as slowly diffusing metastable atoms hit the walls. In "on" cells large numbers of metastables are created during each discharge. In "off" cells they can be created by conditioning pulses that cause discharges about one hundred times less frequently than in the "on" state. If these conditioning pulses are removed and if the exciting voltage across both grids is raised above the firing potential, the observer can, in principle, write directly on the display by means of a light emitting pencil. If he illuminates a group of cells, photo electrons released from the walls initiate the discharges that turn the cells "on." The procedure works very well for a single cell. It has not, however, been tested for the important case when selected series of cells are to be turned on in the neighborhood of many cells that are to stay off. Unless each cell is well shielded optically from its neighbors the first cell to turn on may initiate a wave of state changes that will quickly turn on the entire display.

We are at present studying the properties of small displays with a linear cell density of 40 cells/inch. The photograph, Fig. 4, shows one of these displays in which the sustaining signal is connected to a 3×4 matrix. In the pattern seven cells are in the "on" state and five are in the "off" state. The actual distance between adjacent cells is 0.025 inch. We have also constructed single cells as small as 0.006 inch

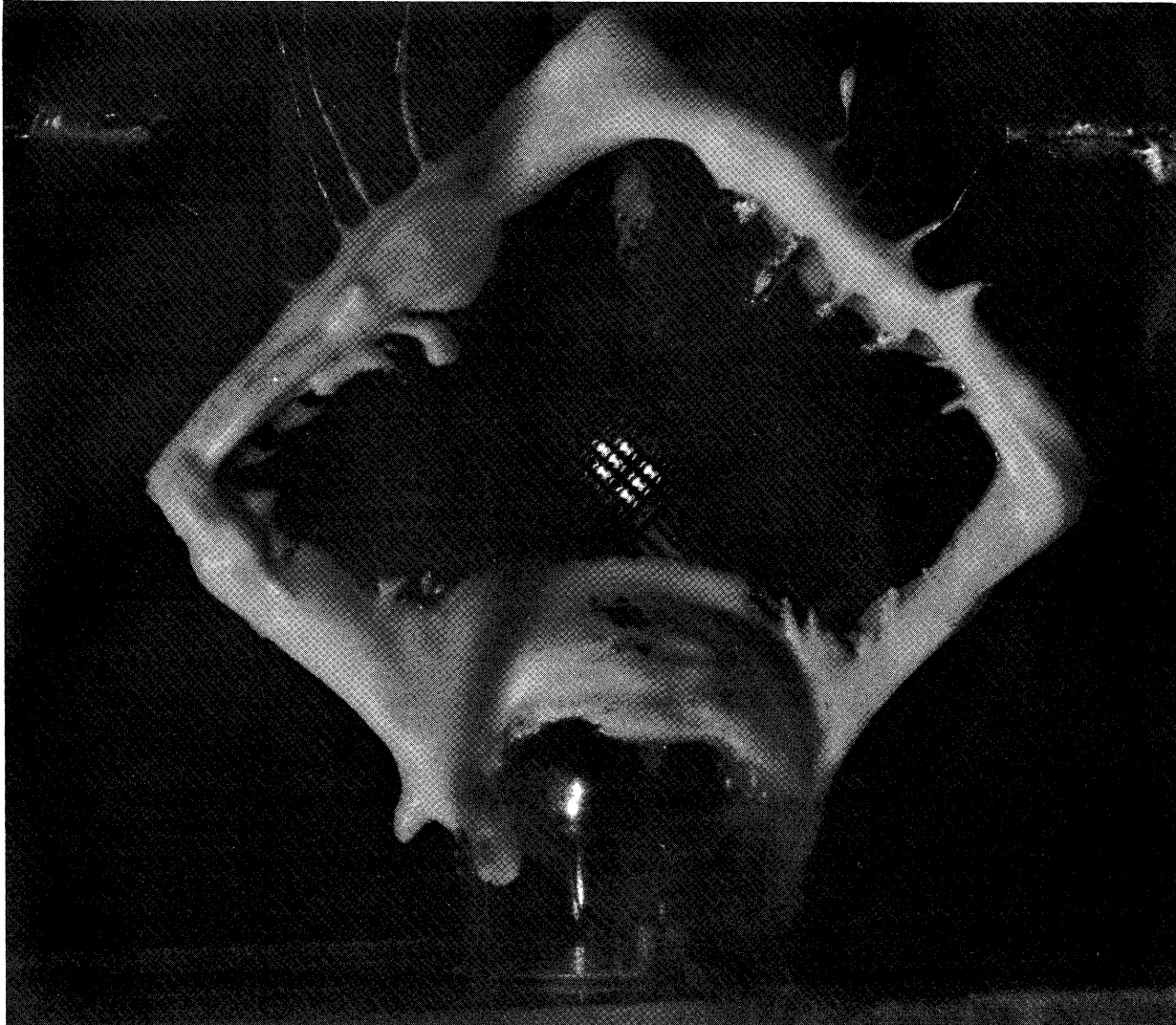


Figure 4. Experimental plasma display.

and we believe that linear densities in excess of 100 cells/inch (10^4 cells/inch²) can be achieved.

To the authors' knowledge no earlier memory displays have utilized pulsed discharges and their associated wall charges. These discharges in larger cells, however, have been observed by a number of investigators, and the influence of the wall charges on the development of these discharges has been understood for many years.⁴⁻⁶ Loeb and El Bakkal in particular have observed that with argon in large cells (diameter = 8 cm) a sequence of pulse discharges could be maintained at 60 cycles/second by a voltage less than that required to initiate the sequence.⁶

SYSTEM PROPERTIES

The development of the Plasma Display Panel was motivated by the anticipated needs of the PLATO computer-based education program at the Coordinated Science Laboratory of the University of Illinois.⁷ The experimental classroom which has been developed as part of this program consists, at present, of twenty student stations, each with a television display and a keyset connected to a central computer (CDC 1604). In addition twelve terminals are available for use at remote locations. For each station the computer transmits signals to a storage tube memory and selects a photographic slide that contains the appropriate text for that stu-

dent. The storage tube and the slide are then scanned simultaneously and the superimposed signals are transmitted to the cathode ray tube.

Within several years there may be similar systems with several hundred stations, and it is not unreasonable to predict that in the future thousands of people in classrooms and even in homes will communicate simultaneously with a central computing facility.

Advances in computer technology have been so rapid that present computers with their high speed, large memory, and steadily decreasing cost per unit operation, are adequate for this kind of service. Display technology, however, has not kept pace with these advances. The TV-storage tube displays, of course, perform well in the PLATO system, and other CRT devices are used successfully in information retrieval systems. However, no display device is now available that performs well, and is sufficiently inexpensive for use in these large systems. If efforts to develop the Plasma Display are successful, this device should meet both performance and cost requirements.

Admittedly, the specific needs of a teaching system emphasize the importance of some properties of a display, and are less demanding of others. Nevertheless, we believe that these needs are similar to the large systems being developed for banks, air line reservations control, and for corporate and university administration. In all of these systems the resolution requirements are at most those met by standard television, information rates are low, and low display cost is imperative. In the remainder of this section we discuss the properties of the Plasma Display (Table I) particularly as they govern the use of this display in these systems. We also discuss briefly its use in the more specialized systems where both high resolution and high data rates are important.

For its use in the display terminals of a teaching system the property of the Plasma Display that most simplifies system design is its inherent memory. Because its images do not need regeneration from an auxiliary memory, information flows from the computer to the display only when the images are changed and then at rates dictated by the uses of the display. Transmission lines can be specified to match these rates, which are much less than the limiting rates acceptable to the display itself.

Experience dictates that in this service a character writing rate of 140 characters/second is completely

satisfactory. This corresponds to a point writing time of 358 microseconds/point, hardly taxing even the slow writing rate on the display. At 30 characters/line and 20 lines/page, an entire page can be written in about four seconds. If the display is equipped with a character generator that sequences the selection of points according to a seven bit code, the corresponding information rate is 1000 bits/second, a rate that is easily accommodated by voice grade telephone lines. Point plotting for maps, graphs, and diagrams is slower. If, for example, we assume a 512×512 raster and specify each point independently by 18 bits, only 55 points are plotted each second. Useful curves, however, can still be plotted in a few seconds. Furthermore with the addition of mode detecting hardware these rates can be increased by a factor of two or three.

Let us assume that each of 3000 remote displays simultaneously receives information at 1000 bits/second from the computer. The information rate on all lines together is then 3×10^6 bits/second, which for a computer with a 48 bit word length calls for one word of output every 16 microseconds, well within the capability of a modern computer. If these 3000 stations are in a single community, each one can be connected over a telephone line to a central distribution point which is in turn connected through a video channel to the computer.

In a teaching system some of the information on the display represents comments or numerical answers entered directly by the student. In some cases he wishes to rewrite all of the information on

Table I
System Properties of Plasma Display Panel

<i>Property</i>	<i>Performance</i>
Cell Density	40-100 cells per inch
Memory	self contained, charge storage on walls
Addressing Mode	digitally addressed
Erase Modes	complete erase, character erase, point erase
Writing and Erase Rate— Slow Mode	10^4 points per second
Writing and Erase Rate— Fast Mode	10^6 points per second
Full Screen Erase Time	5 microseconds
Brightness	comparable to cathode ray tube
Power Input	100 microwatts per point lit

the screen; in other cases he wishes to rewrite only part of the information. Two erase processes have therefore been provided in the PLATO system. One erases the entire screen. The second selectively erases a single character. It would be desirable to extend selective erase to specified points but because of the interaction in the storage tube between the electron beam and the stored charge this cannot now be accomplished. In the Plasma Display every point is addressable. Full erase, and both character and point selective erase, can therefore be implemented.

The resolution requirements of a teaching system are easily met by the Plasma Display. With a cell density of 1600 cells/inch², a 512 × 512 raster corresponds to a display that is 13.8 inches on each side. This provides both adequate viewing area and resolution better than that provided by the television display in the PLATO system.

Although information rates and resolution requirements may be the same in many of these systems, this may not be true for display size and shape. A bank teller, for example, may want to see the name, code number, and account balance of a depositor. Or a purchasing agent may request the name of a company, its quotation on a bid, and perhaps a record of previous business transactions. A strip display with 512 columns but only 64 rows might be entirely appropriate for these applications. For the corporate board room, or for the military command room, much larger displays are indicated.

In this connection it is important to draw another comparison between the Plasma Display and the cathode ray tube. The density of points on a cathode ray tube varies roughly with the size of the tube and the number of resolvable points remain about the same. On the other hand, the number of resolvable points in the Plasma Display can be increased simply by adding more cells. The cell size can also be varied over a range of at least ten to one, but the limits are not yet known. These properties of the display allow considerable freedom in planning large wall displays.

An additional property of the display that may be useful in special applications is that it can be fabricated with curved surfaces.

There are, today, an increasing number of applications that will fully exploit the resolution and fast writing properties of the Plasma Display. The display of dynamic processes, for example, requires video signals, and for the production of high quality

still photographs and motion picture films high resolution is a necessity. Since this service is usually provided by high quality cathode ray tubes whose images are regenerated from magnetic core memories, we compare the appropriate properties of the Plasma Display directly with those of the cathode ray tube.

The maximum writing rate of commercial electrostatic CRT display systems is about 200,000 points/second. This limit is set by the settling time of the digital to analog converters and of the circuits that position the electron beam. If, to avoid flicker, we stipulate 20 frames/second as the minimum frame rate, 10,000 points can be displayed. If the pattern on the screen represents a dynamic process, such as wave motion, and if it changes every frame these rates are actual information flow rates. If, on the other hand, the pattern is stationary these rates are used only to provide the greatest detail possible in a flicker free display.

The storage tube television display of the PLATO system has no flicker problem but its resolution is not as great as the directly addressed electrostatic tube. Magnetically focussed and directly addressed cathode ray tubes offer greater resolution, but at the expense of writing speed.

Because of the inherent memory of the Plasma Display, the number of points in a stationary pattern is limited only by the number of cells in the raster. The limit is much lower when dynamic processes are represented by rapidly changing patterns. If a writing rate of one point/microsecond can be maintained on large displays, a pattern containing 50,000 points can be completely changed at the rate of 20 frames/second.

Two useful features of present cathode ray tube display systems are that information on the display is available to the computer, and that by means of a light pen a programmer can manipulate patterns on the display. These features are also present in the Plasma Display, but it appears that each can be extended. The display itself functions as an auxiliary memory that can be consulted by the computer, and to the ability to manipulate patterns may be added the ability to draw patterns directly on the screen.

A comparison of quality in the two kinds of displays is difficult because the character of the displays is different. In the Plasma Display the number of addressable points and the number of resolvable points are the same. This is not true in the cathode ray tube. It is now possible to address a raster of

4096 × 4096 points, but the number of resolvable points is at best about 2000 × 2000.

At present we cannot realistically consider cell densities greater than 100 cells/inch for the Plasma Display. Thus, the display must be 20 inches wide to match the best cathode ray tube resolution and 40 inches wide to provide 4000 resolvable lines. Except for large displays, the cathode ray tube can provide a picture of higher quality. Furthermore, it is possible to draw continuous lines on the cathode ray tube, and in photographic work this is sometimes important.

In the preceding discussion we have not considered any of the very active research on electroluminescence, or on electron-hole recombination in semiconductors. We believe, however, that it is appropriate to call attention to a different type of gaseous discharge display in which the basic cells are direct current discharges.^{8,9} This display has some of the same advantages as the Plasma Display. To remove addressing ambiguities, however, it seems necessary to isolate the cells from one another by inserting a resistor in the connecting lead to each cell.⁹ The resulting fabrication difficulties appear to limit the achievable cell densities.

CONCLUSIONS

We must emphasize again that the Plasma Display is in an early stage of development. We are working with small matrices, and we have only recently transmitted signals from the computer to change cell states. Nevertheless we know of no fundamental obstacles that will frustrate this development, and if the development is successful the Plasma Display will fill an important role in the computer technology of the near future.

ACKNOWLEDGMENTS

The progress in the development of the Plasma Display depends and will continue to depend on the consultation and assistance of many members of the staff of the Coordinated Science Laboratory. In particular, the authors wish to thank Mr. Brij Arora and Dr. Robert Willson who have contributed on a day by day basis.

REFERENCES

1. D. L. Bitzer, H. G. Slottow, and R. H. Willson, "A Preliminary Description of the C.S.L. Plasma Display," Internal Report—Coordinated Science Laboratory, Univ. of Ill., Urbana, Ill.
2. ———, "Gaseous Display and Memory Apparatus," Application for United States Letters Patent #521,357 dated Jan. 1966.
3. R. H. Willson, "A Capacitively Coupled Bistable Gas Discharge Cell for Computer Controlled Displays," Report R-303, Coordinated Science Laboratory, Univ. of Ill., June 1966.
4. S. Whitehead, *Dielectric Phenomena of Solids*, Clarendon Press, Oxford, 1951, Chap. 4, pp. 171 ff.
5. G. Francis, *Ionization Phenomena in Gases*, Butterworth Publication, London, 1960, Chap. 4.
6. J. M. El Bakkal and L. B. Loeb, "Electrical Breakdown of Argon in Glass Cells with External Electrodes at Constant and at 60-Cycle Alternating Potential," *J. Appl. Physics*, vol. 33, no. 4, (1962).
7. D. L. Bitzer and P. G. Braunfeld, "Description and Use of a Computer Controlled Teaching System," Proc. National Electronics Conference, Oct. 1962, pp. 787-792.
8. Jess J. Josephs, "A Review of Panel-Type Display Devices," Proc. I.R.E. vol. 48, no. 8, (1960).
9. Lear Siegler, Inc., Q.P.R., 2, 3, 4, 5, 6, and 7 "Development of Experimental Gas Discharge Display," Contract Nobsr—89201 Bu Ships, August, 1963-June 1965.

THE USE OF SEMI-RECURSIVE POLYNOMIALS IN THE DESIGN OF NUMERICAL FILTERS

Charles B. Stallings

*Computer Applications Department,
Martin Company, Orlando, Florida*

INTRODUCTION

Numerical filtering, as it applies to processing discrete time series representing missile trajectory data, is usually concerned with the problems of: (1) smoothing; (2) interpolation; and (3) prediction. This paper will be concerned only with the first two of these problems.

The process of smoothing is based on the assumption that the desired information can be separated from the unwanted "noise" on the basis of frequency discrimination. Since the information is assumed to be composed primarily of the low frequency components of the recorded signal, and the "noise" is contrarily composed of the high frequency components of the signal, smoothing becomes a problem of developing an adequate low frequency pass filter.

It is not enough that the output time series of a smoothing filter "look" smooth. If, for example, the output time series represents position data and it needs to be differentiated to obtain velocities and accelerations, the differentiating process acts to amplify the high frequency noise. The result is that the accelerations obtained from doubly differentiating an apparently "smooth" position time series might be very noisy. Therefore, if the resulting accelerations are to be smooth, the filter which smooths the position data must severely attenuate the high frequency noise. An additional requirement of the filter, how-

ever, is that while it severely attenuates the high frequency noise, it must do so without flattening the low frequency information.

This paper will describe a simple method of developing numerical filters which will: (1) preserve all information below a given cut-off frequency; (2) provide a roll-off in the intermediate frequencies of practically any desired steepness; and (3) severely attenuate all high frequency components of a given discrete time series.

THE SEMI-RECURSIVE PARABLOA.

In the discussion immediately following, the letter X will represent the "unsmoothed" or input time series and the letter Y will represent the smoothed or output time series.

Consider Equation (1)

$$Y(t + \Delta t) = W_1 Y(t - \Delta t) + W_2 Y(t) + W_3 X(t + M \Delta t) \quad (1)$$

where $Y(t + \Delta t)$ = the next smooth value to be obtained at time $t + \Delta t$,

$Y(t)$ = the last smooth value obtained at time t ,

$Y(t - \Delta t)$ = the smooth value obtained immediately previous to $Y(t)$ at time $t - \Delta t$,

$X(t + M \Delta t)$ = the unsmoothed "look-ahead" value located at time $t + M \Delta t$

(M is a parameter describing how "far" the process "looks ahead" into the unsmoothed data.)

W_1, W_2, W_3 are the weights chosen such that Eq. (1) is the equivalent of fitting $Y(t - \Delta t)$, $Y(t)$ and $X(t + M \Delta t)$ to a parabola with respect to time and then reading the value $Y(t + \Delta t)$ off of that parabola. Specifically these weights are

$$W_1 = \frac{1 - M}{1 + M}$$

$$W_2 = 2 \frac{M - 1}{M}, \quad M \geq 1$$

$$W_3 = \frac{2}{M(M + 1)}$$

The value M is not restricted to integers. If M is not an integer, the value $X(t + M \Delta t)$ represents some type of average of two (or more) points in the unsmoothed "look-ahead" region.

Using Eq. (1) the process starts at the beginning of the time series with two smooth values (how these "end points" might be chosen will be discussed below under the heading "Considerations Involved in Applications," although it should be pointed out here that, using the techniques described in this paper, it is necessary to obtain only two initial points and two terminal points to solve the end point problem.) Then beginning with the third point, the smoothing process continues through the data series with a

point by point application of Eq. (1), with the value $Y(t + \Delta t)$ determined at one point becoming the value $Y(t)$ used in determining the next smooth value. It is in the use of the two previously smoothed values in conjunction with one unsmoothed value in obtaining $Y(t + \Delta t)$ that Eq. (1) is called a semi-recursive parabolic equation.

It can be said that the smoothing process so described is pursuing the noisy data as represented by the "look-ahead" point $X(t + M \Delta t)$.

The complex transfer function, $P(\omega)$, of the process represented by Eq. (1) is given by

$$P(\omega) = \frac{Y_0}{X_0} = \frac{W_3 e^{jM(\omega\Delta t)}}{e^{j(\omega\Delta t)} - W_1 e^{-j(\omega\Delta t)} - W_2} \quad (2)$$

Since (2) is a complex transfer function, there is associated with this process not only an amplitude response, $R(\omega)$, as a function of the frequency, ω , but an unwanted phase shift $\phi(\omega)$. Figures 1 and 2 are plots of $R(\omega)$ vs. ω and $\phi(\omega)$ vs. ω respectively.

The simplest method of eliminating this phase shift is to run the smooth time series resulting from the use of Eq. (1) through the same process, only "backwards" with respect to time using the equation,

$$Y(t - \Delta t) = W_1 Y(t + \Delta t) + W_2 Y(t) + W_3 X(t - M \Delta t) \quad (3)$$

where here the letter X represents the output data series resulting from the use of Eq. (1).

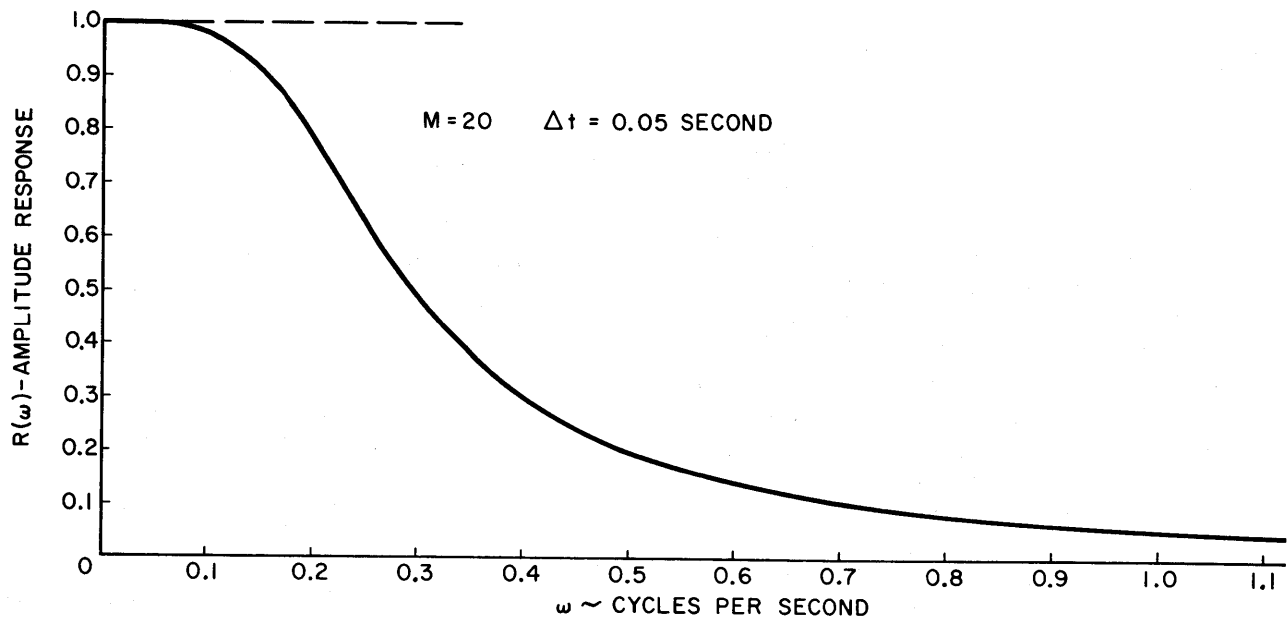


Figure 1. Amplitude response vs ω of recursive parabolic Eqs. (1) and (2).

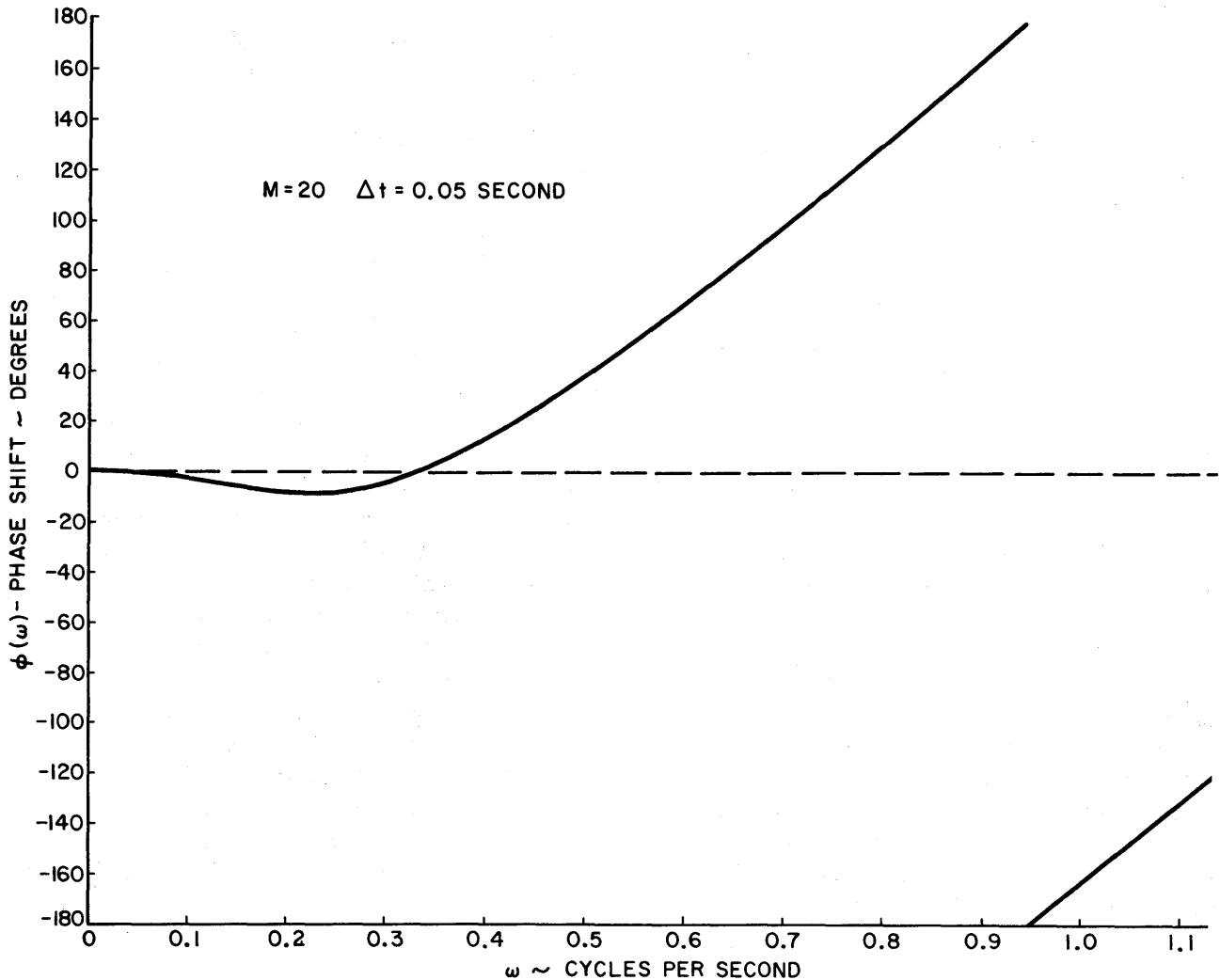


Figure 2. Phase shift vs ω of recursive parabolic Eq. (1).

The complex transfer function, $P^*(\omega)$, of the process represented by Eq. (3) is obviously the complex conjugate of the complex transfer function, $P(\omega)$, of the process represented by Eq. (1). Therefore, the transfer function, P_1 , of the combination of first processing the original time series by Eq. (1) and then processing the resulting series by Eq. (3) is

$$P_1 = [P(\omega)][P^*(\omega)] = |P(\omega)|^2 \quad (4)$$

This is represented in block diagram form in Fig. 3(a) and since P_1 is real for all ω and has no phase shift associated with it, the plot of amplitude response, $R(\omega)$, vs. ω given in Fig. 4 reflects the complete nature of this filter arrangement.

The frequency response of the filters discussed here is determined by the size of the parameter, M , and the size of the sampling interval, Δt , in Eqs.

(1) and (3). Since the size of the parameter M determines how many data points the process "looks ahead" into the noisy data, it is obvious that if M is small the process can follow closely the short term changes in the data—which is the same as saying that it has a good high frequency response. If, however, M is large, the process will respond primarily to the longer-term changes, and the higher frequencies will be more severely attenuated.

For the purposes of this paper, when using Eqs. (1) and (3), values of $M = 20$ and $\Delta t = 0.05$ second will be used except when otherwise specified.

While the filter P_1 (Fig. 3(a)) has fair smoothing properties, it is still far from ideal. First, for any cut-off frequency $\omega_c > 0$, arbitrarily chosen, there is associated with it a certain amount of amplitude attenuation or information loss. For example, if $\omega_c =$

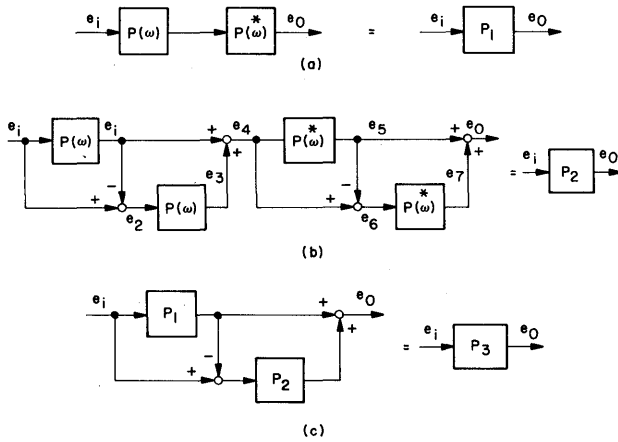


Figure 3. Sequence of operations to obtain filters from the basic recursive parabolic Eqs. (1) and (3).

0.15 is arbitrarily chosen as the cut-off frequency, filter P_1 will cut the signal amplitude at $\omega = \omega_c$ to approximately 91.5% of their original value and in the rest of the low frequency information region, $0 \leq \omega \leq \omega_c$, there is some degradation of signal. In the second place, if the resulting smooth data is to be differentiated, the attenuation of the high frequency noise is probably inadequate. A possible solution to the second problem is to run the results of one pass through filter P_1 , through P_1 again, making successive passes in this fashion until the high frequency components have been sufficiently attenuated.

However, this solution of the high frequency or noise problem would further attenuate the low frequency ($0 \leq \omega \leq \omega_c$) information.

To solve this problem a filter is needed in which the amplitude response curve remains flat at $R(\omega) = 1.0$ in the region $0 \leq \omega \leq \omega_c$, and then falls off in the region $\omega > \omega_c$. Examine the sequence of operations leading to filter P_2 , block diagrammed in Fig. 3(b). Here, the original time series e_i is smoothed using Eq. (1) to obtain the time series e_1 . Then we subtract e_1 from e_i to obtain the residual time series e_2 . e_2 now contains all the high frequency noise removed from e_i in obtaining e_1 . Also contained in e_2 is the portion of the low frequency information ($0 \leq \omega \leq \omega_c$) lost in obtaining e_1 . Since it is only this low frequency information that we are interested in, e_2 is smoothed by Eq. (1) to obtain the time series e_3 which then contains primarily that part of the low frequency information which was lost in obtaining e_1 . By adding e_3 to e_1 we obtain the time series e_4 , which contains almost all of the low frequency information. However, since there is associated with this process of obtaining e_4 an unwanted phase shift, e_4 is now run through the same type of sequence of operations, only by using Eq. (3) instead of Eq. (1) the phase shift is reversed so that the final output time series e_0 of filter P_2 is in phase with e_i for all ω .

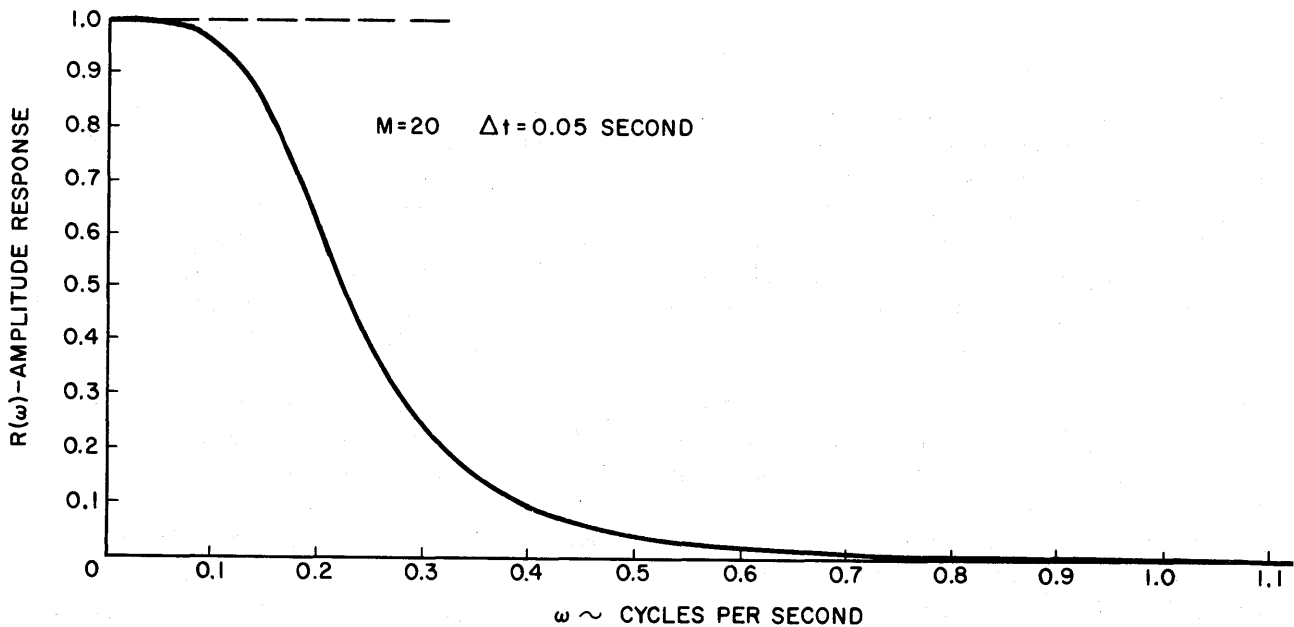


Figure 4. Amplitude response vs ω of filter P_1 (Fig. 3a).

The resulting real transfer function of the filter P_2 is given by

$$P_2 = \{2P(\omega) - [P(\omega)]^2\} \{2P^*(\omega) - [P^*(\omega)]^2\} = \frac{|2P(\omega) - [P(\omega)]^2|^2}{|2P(\omega) - [P(\omega)]^2|^2} \quad (5)$$

Notice that in the sequence of operations used to obtain filter P_2 , subtractions of complex signals which are in general out of phase with each other are performed (first when subtracting e_1 from e_i to obtain e_2 and then when subtracting e_5 from e_4 to obtain e_6). This implies that in general, the absolute magnitude of any given frequency component in e_2 (or e_6) will be greater than the difference between the absolute magnitudes of the same frequency component contained in e_i and e_1 (or e_4 and e_5), or

$$\begin{aligned} e_2 &= |e_i - e_1| \geq |e_i| - |e_1| \\ \text{and } e_6 &= |e_4 - e_5| \geq |e_4| - |e_5| \end{aligned} \quad (6)$$

(Schwartz's inequality).

In the low frequency region this increase in signal amplitude of the residual time series e_2 and e_6 almost exactly compensates for the subsequent attenuation which results, first from the smoothing process used in obtaining e_3 and e_7 and, second, from the out-of-phase additions performed in obtaining e_4 and e_{10} . As a result the plot of amplitude response, $R(\omega)$, vs. ω of filter P_2 is close to the ideal mentioned above, as can be seen in Fig. 5. Here the value of $R(\omega)$ in the region $0 \leq \omega \leq \omega_c$ is approximately constant at 1.0 and then falls off rather

sharply as ω increases above ω_c . It should be mentioned here that for all the low pass filters developed using these techniques, the amplitude response $R(\omega)$ monotonically decreases as ω increases in the region from ω_c to ω_n (ω_n = the folding frequency characteristic of the sampling rate = $1/(2\Delta t)$). The lack of such a monotonic decrease of $R(\omega)$ is one of the more objectionable properties of many numerical filters. The maximum distortion from the value $R(\omega) = 1.0$ in the region $0 \leq \omega \leq \omega_c$ is approximately +0.3%. ω_c the cut-off frequency has an approximate value of 0.15. As indicated above ω_c is a function of M and Δt . A good approximate relationship between ω_c , M and Δt for filter P_2 (and P_3 as developed below) is,

$$\omega_c = \frac{0.15}{M\Delta t} \quad (7)$$

With a filter such as P_2 it is possible to severely attenuate the high frequency noise without distorting the low frequency information excessively. If the high frequency noise is not attenuated severely enough by one pass through filter P_2 , the output of the first pass can be fed back through filter P_2 . This process can be repeated until the user considers the high frequency noise adequately attenuated, or until he considers the low frequency ($0 \leq \omega \leq \omega_c$) distortion too severe. (After 10 successive passes through P_2 the low frequency distortion from the ideal of $R(\omega) = 1.0$ would be approximately 3%).

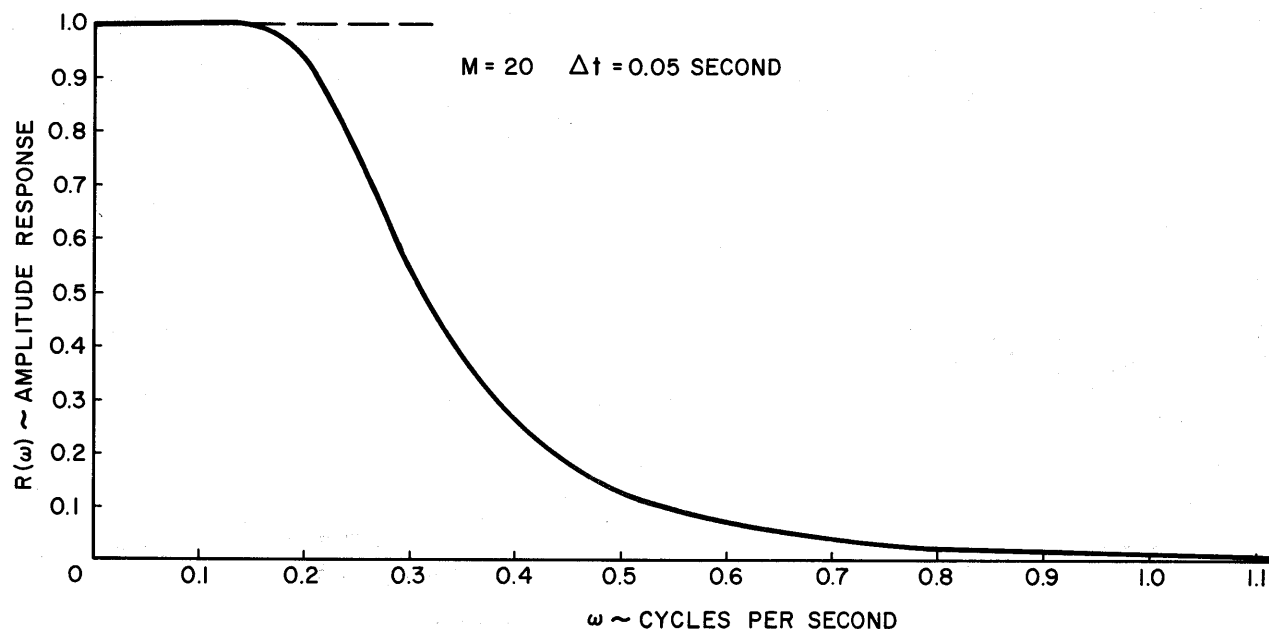


Figure 5. Amplitude response vs ω of filter P_2 (Fig. 3b).

Consider next the sequence of operation used to obtain filter P_3 , Fig. 3(a). Here filter P_1 is used to smooth e_i and P_2 is used to smooth the residual time series which results from subtracting the smoothed output of P_1 from e_i . The output of P_2 is then added back into the output of P_1 to obtain e_o . The resulting real transfer function of this sequence of operations is given by

$$P_3 = P_1 + P_2 - P_1 P_2 \quad (8)$$

The plot of amplitude response, $R(\omega)$, vs. ω is shown in Fig. 6. The maximum distortion of the amplitude response from $R(\omega) = 1.0$ of filter P_3 in the low frequency ($0 \leq \omega \leq \omega_c$) region is approximately +.03%.

Filter P_3 has been used with a high degree of success in smoothing noisy radar position data. After 10 successive passes through P_3 the position data was smooth enough to yield smooth accelerations by differentiating twice using only 3 point parabolic differentiation.

For those who are accustomed to the language of communication engineering, the degradation from $\omega_c = 0.15$ cps to $\omega = 0.3$ cps of 10 passes through filter P_3 is approximately 36 decibels. Figure 6 also shows a plot of amplitude response $R(\omega)$ vs. ω of 10 successive passes through filter P_3 . The maximum low frequency distortion from $R(\omega) = 1.0$ is approximately = 0.3%.

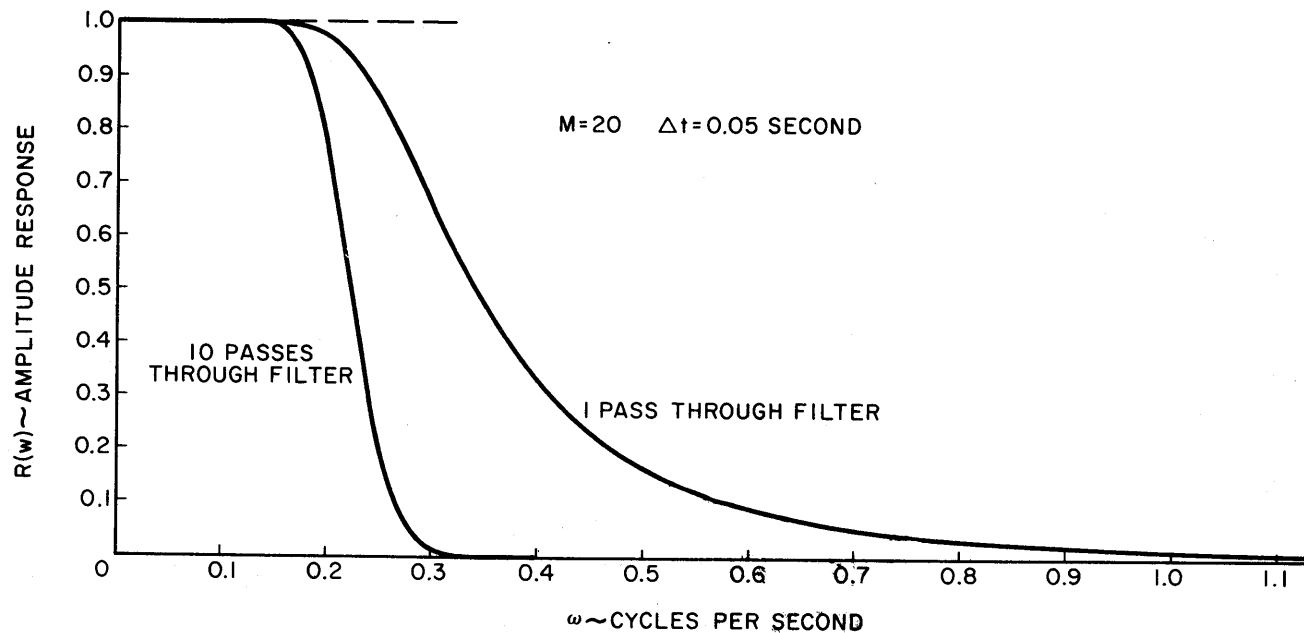


Figure 6. Amplitude response vs ω of filter P_3 (Fig. 3c).

CONSIDERATIONS INVOLVED IN APPLICATIONS

End points: As pointed out above, the end point problem using filters developed from Eqs. (1) and (3) is solved by obtaining two smooth initial values for Eq. (1) and two smooth terminal values for Eq. (3). Any of several standard methods can be used to obtain these four end points. For example, one method which has proven satisfactory in many cases is to fit a parabola to the first (or last) K points (where K is large enough to suppress noise) of the time series, and then read the first (or last) two points off of this parabola. In other cases the first (or last) two points have been chosen by visual judgment. The important consideration is that these first (or last) two points are consistent enough with each other to give a reliable initial (or terminal) slope for the smooth data. Once good end points are obtained, they should be retained even for subsequent passes through the filter.

Interpolation: The techniques using Eqs. (1) and (3) lend themselves very conveniently to interpolating for points to replace missing or bad points. If, for example, in the first pass through a filter the "look-ahead" parameter has a nominal value of $M = 20$ and there are 5 consecutive bad points in the area immediately in front of the "look-ahead" point, the value of M can then be increased to 25, thereby jumping over the 5 bad points. After pass-

ing this area M then resumes its nominal value of 20. This change in M is necessary for the first pass through the filter only. The result will be a set of 5 consistent (with respect to the rest of the smooth data) interpolated values.

Variable frequency response: Since the frequency response of equations (1) and (3) given a fixed Δt is a function of the "look-ahead" parameter M , all that is needed to modify the frequency response in smoothing a given time series using filter P_3 , is to vary M . Compare Fig. 6 with Fig. 7. Both of these plots represent a plot of $R(\omega)$ vs. ω of filter P_3 . The difference is that in the case represented by Fig. 6, $M = 20$, while in the case represented by Fig. 7, $M = 40$.

When using a variable frequency response on a given time series, it is obvious from the nature of Eqs. (1) and (3) that both position (in the case when the time series represents position vs. time) and the velocities obtained from the resulting smooth time series will be continuous at the junction times where M changes. However, the second derivatives or accelerations will tend to be discontinuous at such junction points. Since these discontinuities in acceleration represent short periods of high frequency distortion, they can be removed by making several more passes through the filter using a constant small value of M . In these additional passes M might assume a value equal to the smallest M used in the previous variable frequency smoothing passes.

Other Filter Design

While the filter P_3 has proven adequate in smoothing radar trajectory data, its value might prove questionable in other applications. In some cases, such as vibration analysis, a high resolution filter to separate close frequencies might be needed. If filter P_3 were used, the many passes through the filter that would be required to obtain such high resolution might result not only in too much distortion in the low frequency pass region ($0 \leq \omega \leq \omega_c$), but also might use an excessive amount of machine time. Examine Fig. 8. This plot shows a high resolution in separating relatively close frequencies, but it takes 80 passes through filter P_3 to obtain such resolution, and the maximum distortion in the low pass region is approximately +2.5%. In view of these inadequacies, other filter designs must be sought.

Consider the semi-recursive cubic equation pair, Eqs. (9) and (10),

$$Y(t + \Delta t) = W_1 Y(t - M\Delta t) + W_2 Y(t - \Delta t) + W_3 Y(t) + W_4 X(t + M\Delta t) \quad (9)$$

$$Y(t - \Delta t) = W_1 Y(t + M\Delta t) + W_2 Y(t + \Delta t) + W_3 Y(t) + W_4 X(t + M\Delta t) \quad (10)$$

corresponding to the semi-recursive parabolic equation pair, Eqs. (1) and (3) respectively. As in the case of Eqs. (1) and (3) the last two smoothed values obtained and an unsmoothed "look-ahead" value are used to obtain the next smooth value, but

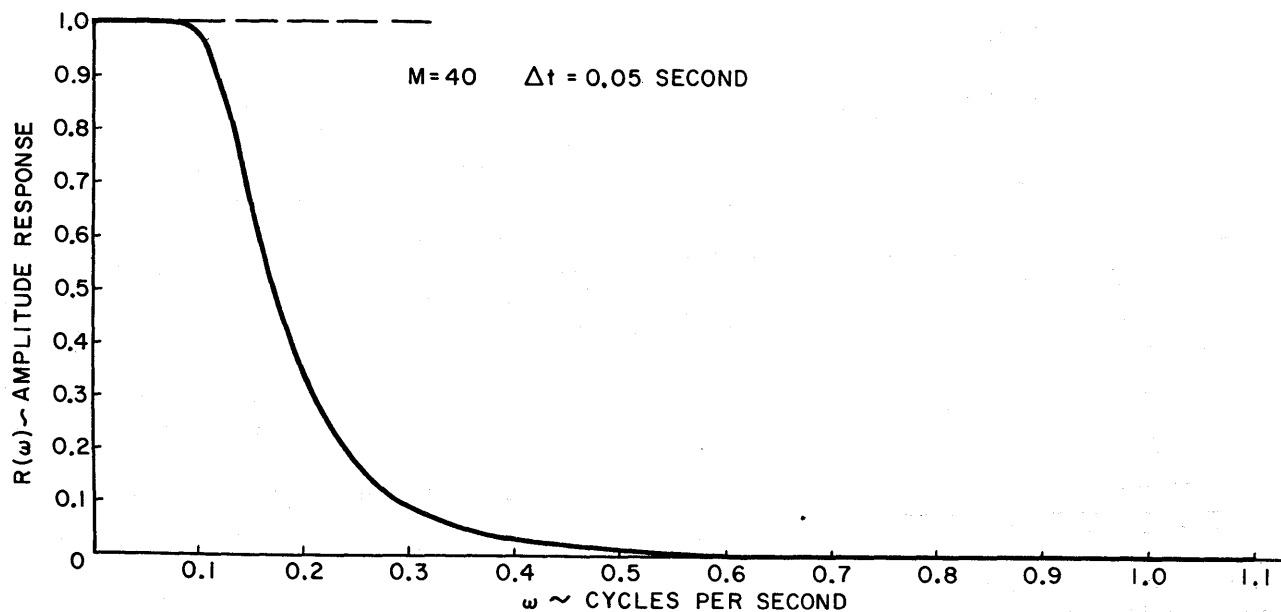


Figure 7. Amplitude response vs ω of filter P_3 (Fig. 3c).

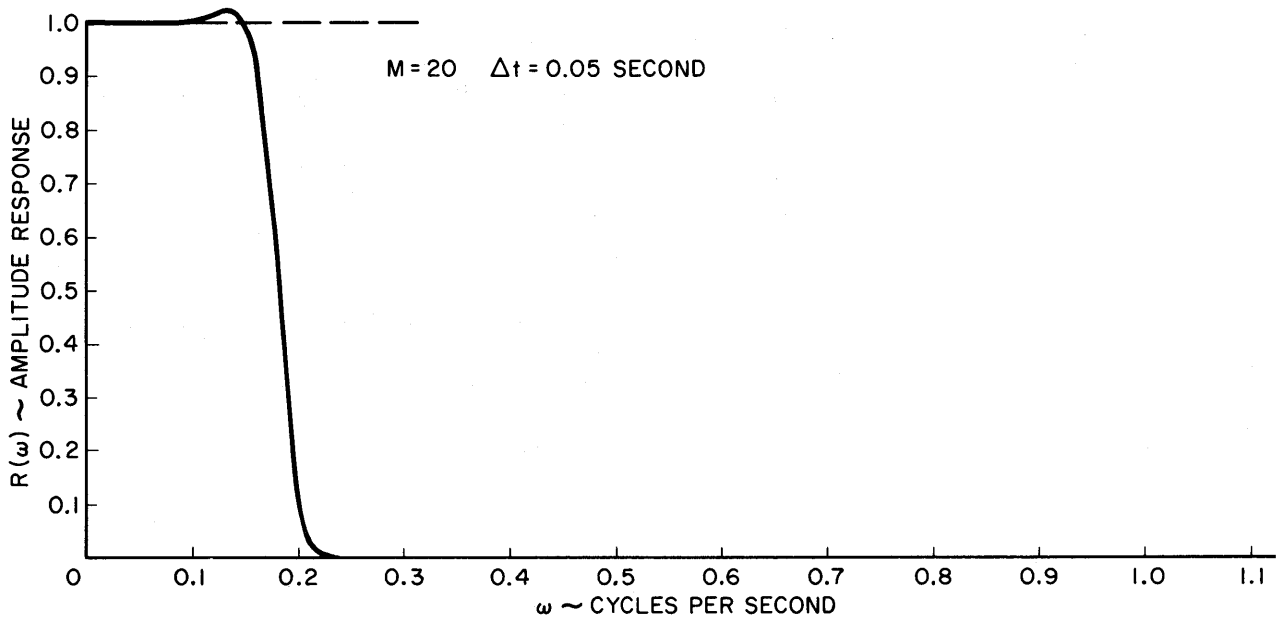


Figure 8. Figure 8 amplitude response vs ω of 80 passes through filter P_3 .

in Eqs. (9) and (10) an additional smooth "look-back" value is included in the evaluation with W_1, W_2, W_3, W_4 chosen such that Eq. (9) is the equivalent of fitting $Y(t - M\Delta t), Y(t - \Delta t), Y(t)$, and $X(t + M\Delta t)$ to a cubic with respect to time and then reading the value of $Y(t + \Delta t)$ off of this cubic. Specifically, these weights are:

$$\begin{aligned} W_1 &= \frac{1}{M^2} \\ W_2 &= -1 \\ W_3 &= 2\left(1 - \frac{1}{M^2}\right) \\ W_4 &= \frac{1}{M^2} \end{aligned}$$

The value of $Y(t - \Delta t)$ in Eq. (10) is obtained in an analogous manner.

The complex transfer function, $T(\omega)$, of the process represented by Eq. (9) is given by

$$\begin{aligned} T(\omega) &= \frac{Y_0}{X_0} \\ &= \frac{W_4 e^{jM(\omega\Delta t)}}{e^{j(\omega\Delta t)} - W_1 e^{-jM(\omega\Delta t)} - W_2 e^{-j(\omega\Delta t)} - W_3} \quad (11) \end{aligned}$$

and the transfer function, $T^*(\omega)$, of the process represented by Eq. (10) is the complex conjugate of Eq. (11).

Figure 9 is a plot of $R(\omega)$ vs. ω of this semi-recursive cubic with $M = 30$ and $\Delta t = 0.05$ second. Also associated with this semi-recursive cubic is an unwanted phase shift, $\varnothing(\omega)$, but as in the case of the semi-recursive parabola, this phase shift can be eliminated by following the first pass across the unsmooth time series using Eq. (9) with a second pass across the resulting smooth time series, "backwards" with respect to time using Eq. (10). This process is represented in block diagram form in Fig. 10 (a). The real transfer function of the resulting filter T_1 is given by,

$$T_1 = [T(\omega)][T^*(\omega)] = |T(\omega)|^2 \quad (12)$$

Figure 11 is a plot of $R(\omega)$ vs. ω of filter T_1 . Compare this curve with the analogous one for filter P_1 shown in Fig. 4. (To make the cut-off frequency, ω_c , of the filters developed from Eqs. (9) and (10) compatible with the cut-off frequency of the filters developed from Eqs. (1) and (3), an $M = 30$ is used here instead of $M = 20$). The roll-off in the region $\omega < \omega_c$ is much sharper, but the distortion in the region $0 \leq \omega \leq \omega_c$ consists of a resonant peak of $R(\omega) = 1.10$, for a maximum distortion of approximately +10%. However, by processes already explained, we can shape this low frequency ($0 \leq \omega \leq \omega_c$) region to suit our use. In the sequence of operations shown in Fig. 10 (b), the input time series, e_i , is first smoothed by filter T_1 to obtain e_1 . Then e_1 is subtracted from e_i to obtain the residual time series e_2 . Next e_2 is smoothed by filter T_1

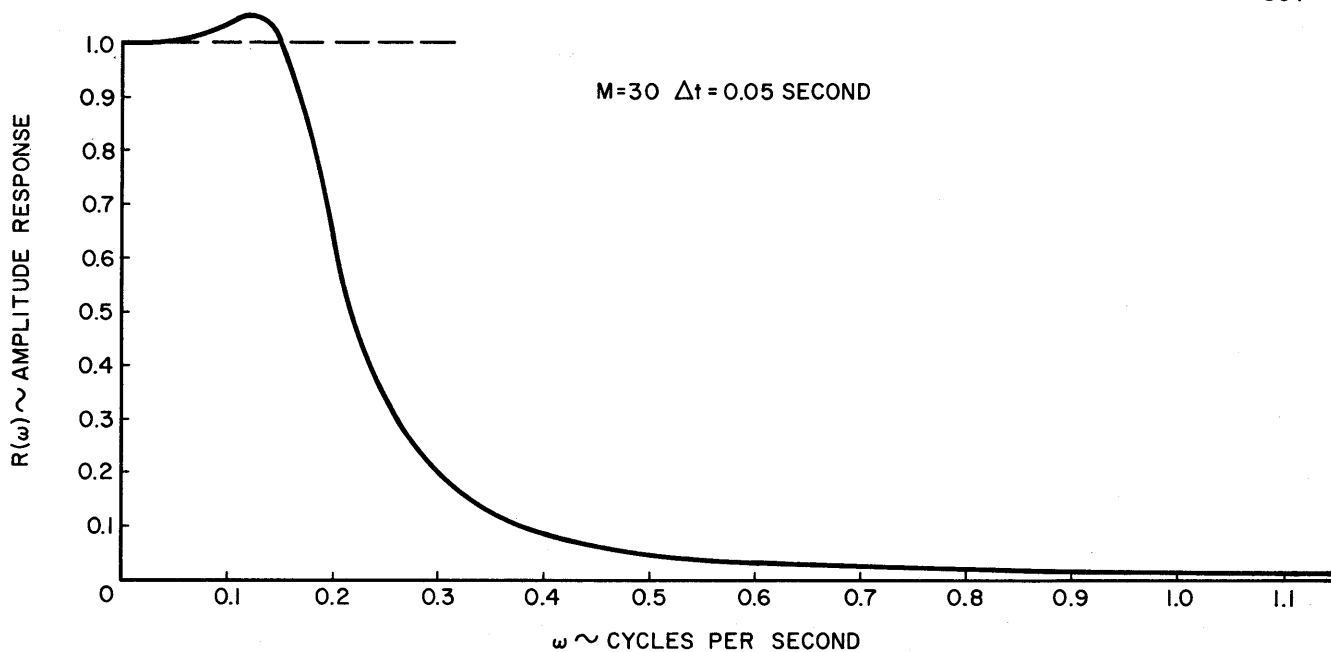


Figure 9. Amplitude response vs ω of recursive cubic equations.

and the resulting time series is added to e_1 to obtain e_0 . This sequence of operations defines filter T_2 and its real transfer function is given by

$$T_2 = 2T_1 - T_1^2 \quad (13)$$

Figure 12 shows the plot of $R(\omega)$ vs. ω of filter T_2 . The maximum distortion from $R(\omega) = 1.0$ in the low frequency ($0 \leq \omega \leq \omega_c$) region is approximately -1.0% .

Now examine Fig. 10 (c). The sequence of operations indicated here is to take the output of filter T_2 and feed it back through filter T_2 for 9 more successive passes (for a total of 10 passes through filter T_2). Figure 13 is a plot of $R(\omega)$ vs. ω of this sequence. Notice that the distortion from $R(\omega) = 1.0$ in the region, $0 \leq \omega \leq \omega_c$ has a maximum of approximately -10% , and compare this with the approximate $+10\%$ distortion associated with filter T_1 . By taking the output of 10 passes through filter T_2 , and passing it through T_1 , we not only remove most of the distortion in the region $0 \leq \omega \leq \omega_c$, but achieve a still sharper roll-off in the region $0 \leq \omega \leq \omega_c$. This process describes filter T_3 with a real transfer function given by

$$T_3 = T_1 (T_2)^{10} \quad (14)$$

Figure 14 is a plot of $R(\omega)$ vs. ω of filter T_3 . The maximum distortion from $R(\omega) = 1.0$ in the region $0 \leq \omega \leq \omega_c$ is approximately $+2.5\%$.

Lastly, by the sequence of operations indicated in

Fig. 10(d) we obtain filter T_4 with a real transfer function given by

$$T_4 = 2T_3 - T_3^2 \quad (15)$$

Figure 15 is a plot of $R(\omega)$ vs. ω of this filter. Compare this with Fig. 8. The maximum distortion from $R(\omega) = 1.0$ in the region $0 \leq \omega \leq \omega_c$ in Fig. 15 is -0.06% compared to a maximum of 2.5% in Fig. 8, and the roll-off in the region $\omega < \omega_c$ is

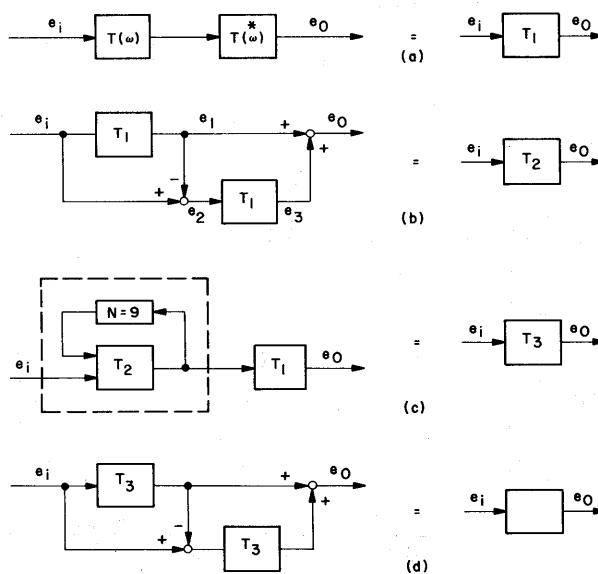


Figure 10. Sequence of operations to obtain filters from the basic recursive cubic Eqs. (9) and (10).

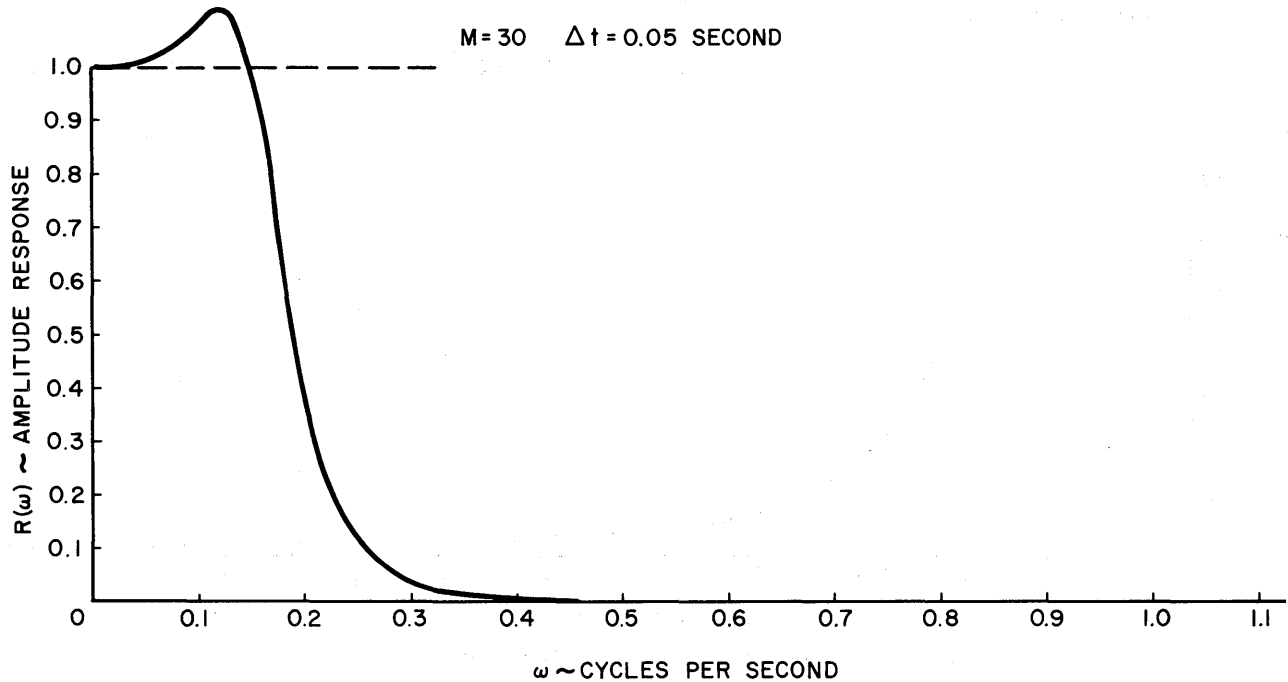


Figure 11. Amplitude response vs ω of filter T_1 (Fig. 10a).

sharper, yet to process a given time series using filter T_4 takes $\frac{1}{4}$ the machine time that it would take to process the same time series by the 80 passes through P_3 that are required to get the frequency response indicated by Fig. 8.

To give those who are accustomed to the language of communication engineering an idea of how sharp this filter T_4 is, the degradation from $\omega_c = 0.15$ cps to $\omega = 0.3$ cps representing one octave is over 244 decibels.

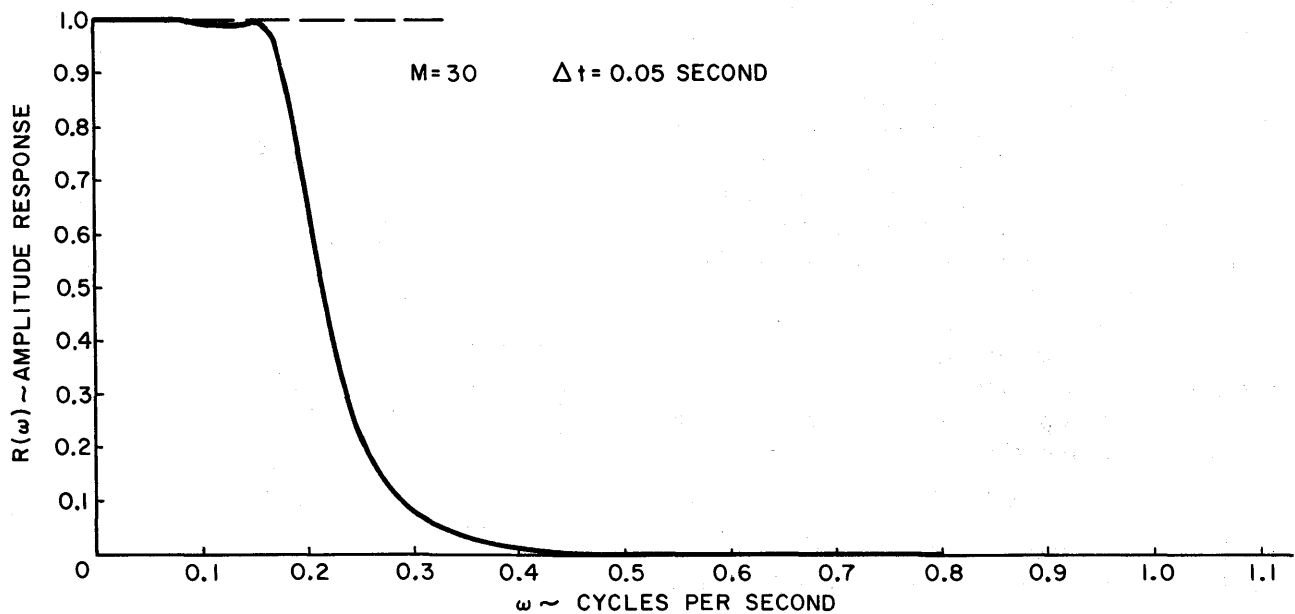


Figure 12. Amplitude response vs ω of filter T_2 (Fig. 10b).

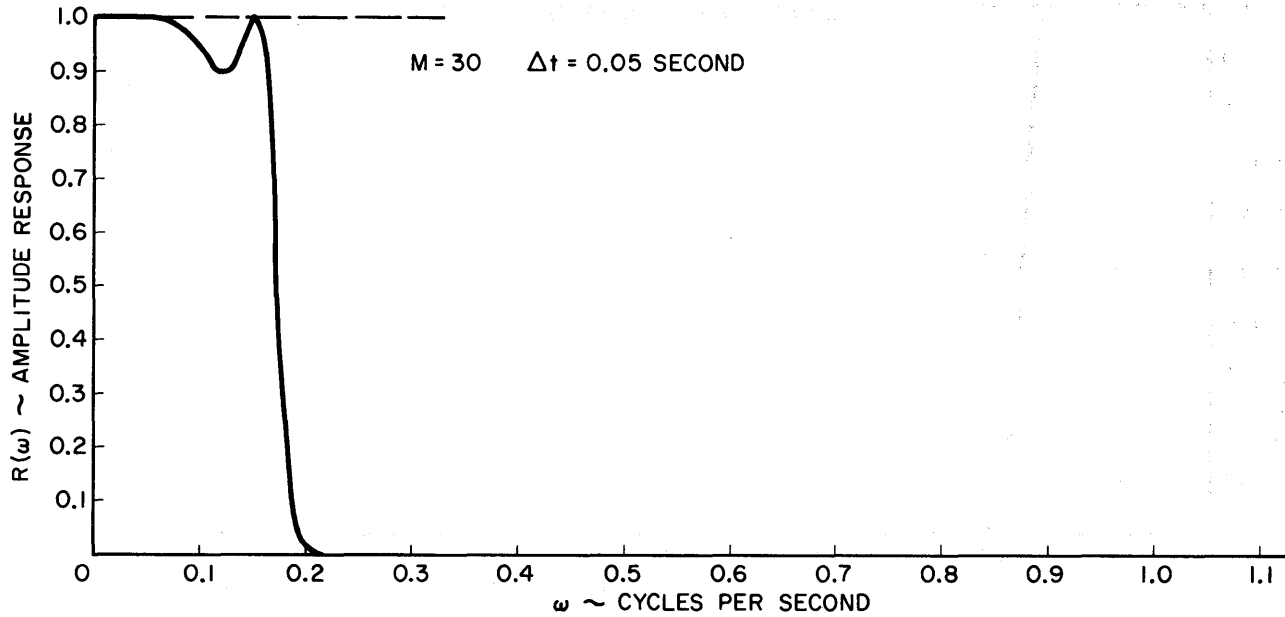


Figure 13. Amplitude response vs ω of 10 passes through filter T_2 .

Band Pass Filters

Once a sharp cut-off low pass filter such as T_4 has been developed it is a small and obvious step to develop a sharp cut-off band pass filter. Figure 16 shows the sequence of operations needed to develop such a filter. First the input time series is processed by filter T_4 using an $M = 30$. The resulting time

series e_1 is then subtracted from e_i to obtain e_2 . e_2 would then contain all of the frequency components in e_i above approximately $\omega = 0.21$. Then e_2 is processed by T_4 using an $M = 15$ to cut-off all frequencies above approximately $\omega = 0.31$. The resulting arrangement is filter T_5 , a band pass filter passing all frequencies in the region $0.21 \leq \omega \leq 0.31$. Fig. 17 is a plot of $R(\omega)$ vs. ω of this filter.

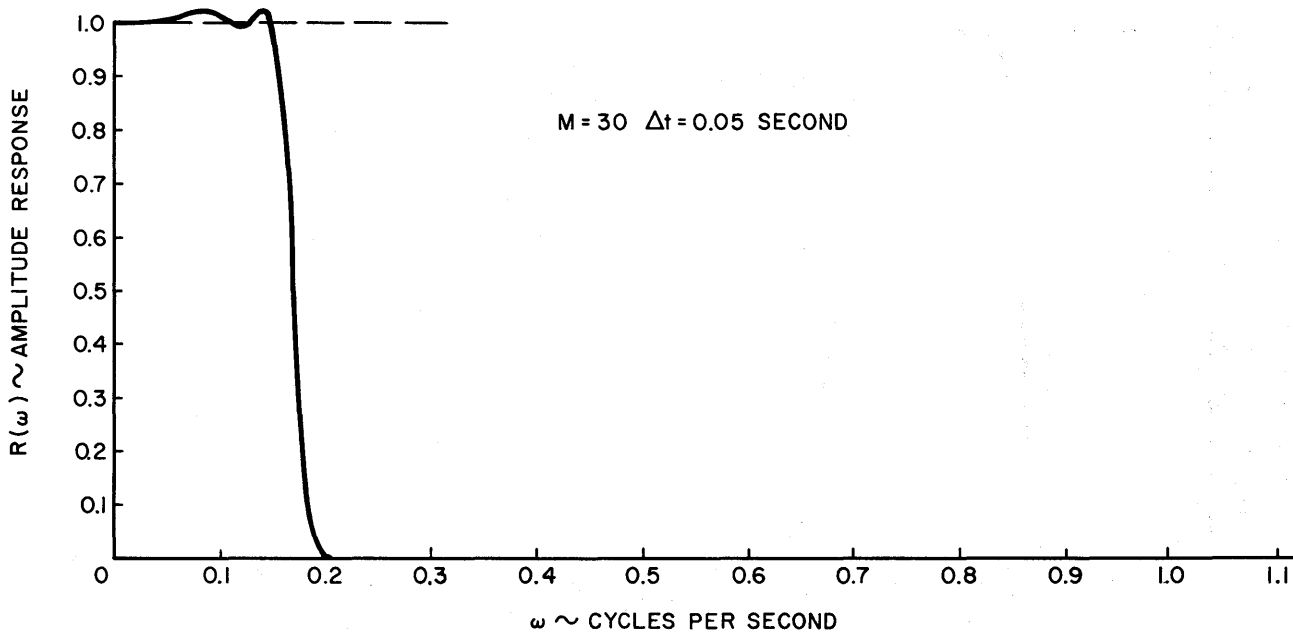


Figure 14. Amplitude response vs ω of filter T_3 (Fig. 10c).

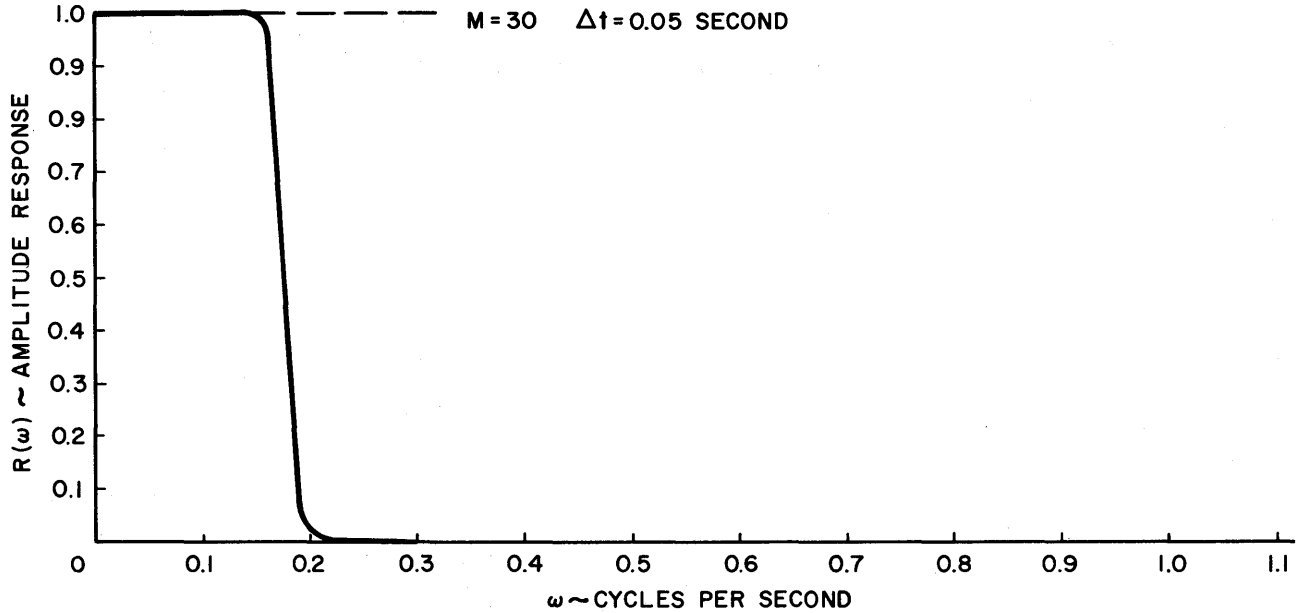


Figure 15. Amplitude response vs ω of filter T_4 (Fig. 10d).

Examples of actual application:

Given the signal

$$X = \sin(\omega_1 t) + \sin(\omega_2 t) + 10 \sin(\omega_3 t) \quad (16)$$

where

$$\omega_1 = 5 \text{ cps}$$

$$\omega_2 = 7.75 \text{ cps}$$

$$\omega_3 \cong 159,150 \text{ cps}$$

$$0.0 \text{ second} \leq t \leq 10.0 \text{ second}$$

$$\text{and } \Delta t = 0.01 \text{ second}$$

Filter P_3 was used in an attempt pass the $\omega_1 = 5$ while eliminating the other two components. Since it is impossible to represent a frequency as high as ω_3

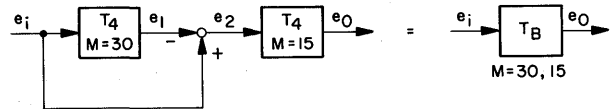


Figure 16. Sequence of operations to obtain a band pass filter.

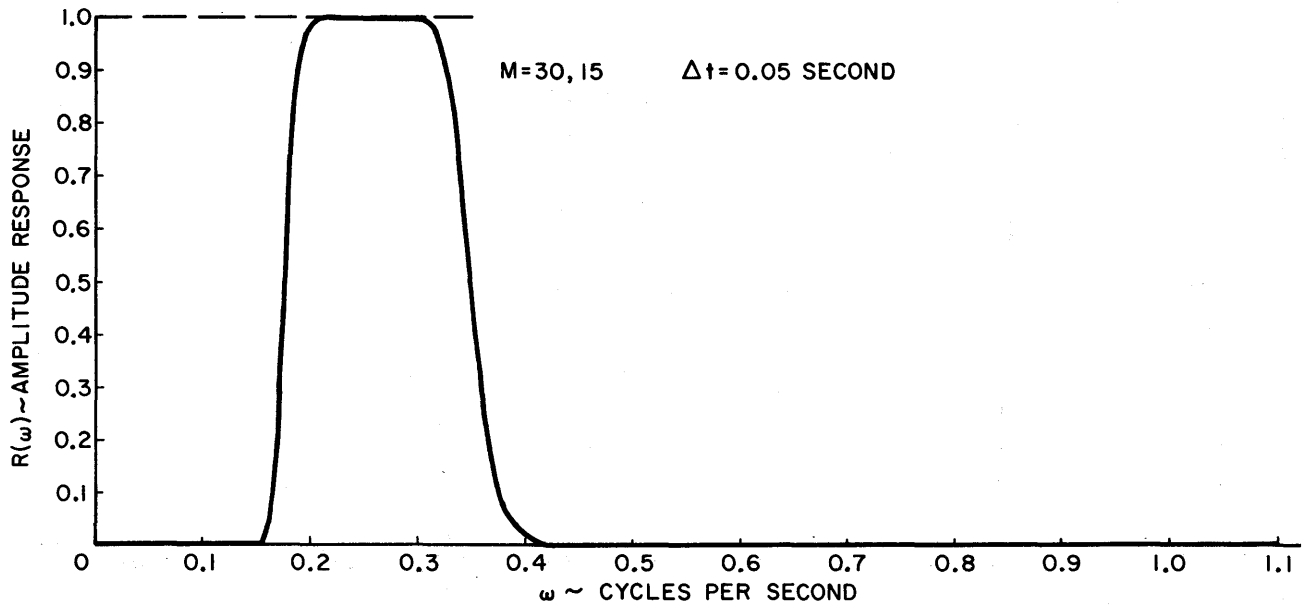


Figure 17. Amplitude response vs ω of filter T_5 (Fig. 16).

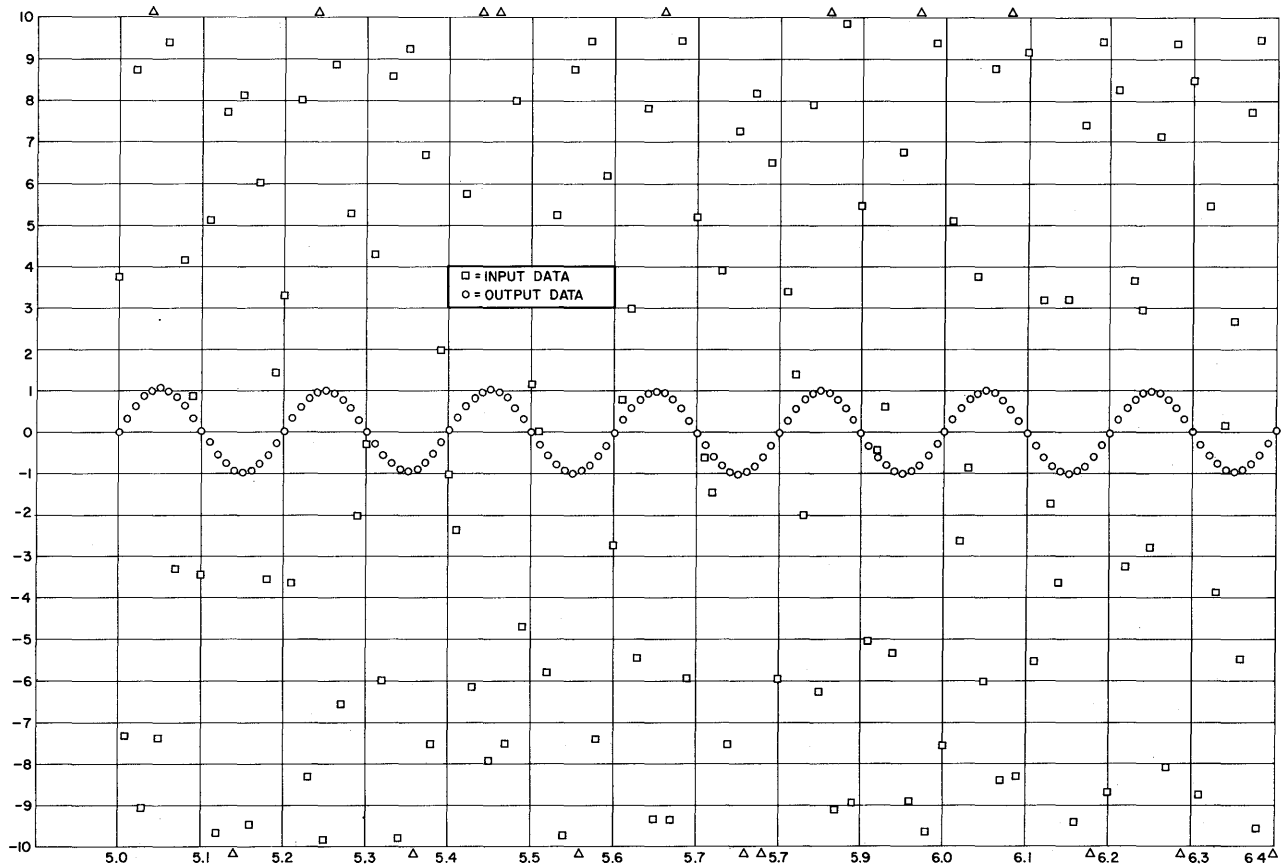


Figure 18. Plot of input and output data.

with a $\Delta t = 0.01$ second, this component was included to simulate high amplitude background "scatter." Examine Fig. 18. In this graph we have the segment of the input signal (Eq. (16)) represented by a square symbol. (The symbol " Δ " represents input points whose amplitudes exceed the graphical limits). The output of 80 passes through filter P_3 using an $M = 3$ is given by the symbol "O". This output is the desired

$$Y = \sin(\omega_1 t), \omega_1 = 5. \text{ cps}$$

What little distortion there is in the output signal

is mostly the result of the low frequency component induced by the aliasing associated with trying to represent ω_3 by a sampling rate of 100 points per second.

ACKNOWLEDGMENT

The author would like to acknowledge a great debt to Mr. Mark Robinson, Numerical Analyst at Martin Company, both for his encouragement and for the time he spent in discussions with the author pertaining to the techniques described in this paper.

FAST FOURIER TRANSFORMS—FOR FUN AND PROFIT

W. M. Gentleman

*Bell Telephone Laboratories
Murray Hill, New Jersey*

and

G. Sande *

*Princeton University
Princeton, New Jersey*

IMPLEMENTING FAST FOURIER TRANSFORMS

Definition and Elementary Properties of Fourier Transforms

The “Fast Fourier Transform” has now been widely known for about a year. During that time it has had a major effect on several areas of computing, the most striking example being techniques of numerical convolution, which have been completely revolutionized. What exactly is the “Fast Fourier Transform”?

In fact, the Fast Fourier Transform is nothing more than an algorithm whereby, for appropriate length sequences, the finite discrete Fourier transform of the sequence may be computed much more rapidly than by other available algorithms. The properties and uses of the finite discrete Fourier transform,

*This work made use of computer facilities supported in part by National Science Foundation grant NSF-GP579. Research was partially supported by the Office of Naval Research under contract Nonr 1858(05) and by the National Research Council of Canada.

which become practical when used with the fast algorithm, make the technique important. Let us therefore first consider some properties of such transforms.

The usual infinite Fourier integral transform is well known and widely used—the physicist when solving a partial differential equation, the communication engineer looking at noise and the statistician studying distributions may all resort to Fourier transforms, not only because the mathematics may be simplified, but because nature itself is often easier to understand in terms of frequency. What is less well known is that many of the properties of the usual Fourier transform also hold, perhaps with slight modification, for the Fourier transform defined on finite, equispaced, discrete sequences.

We see in Table I that the most significant change required to modify the usual theorems so that they apply to the finite discrete case is that indexing must be considered modulo N . A useful heuristic interpretation of this is to think of the sequence $X(t)$ as a function defined at equispaced points on a circle. This interpretation is to be contrasted with what is

TABLE I

A Comparison of Usual and Finite Discrete Fourier Transforms

Usual	Finite Discrete
Definition	
$\hat{X}(\hat{t}) = \int_{-\infty}^{\infty} X(t)e^{2\pi i t \hat{t}} dt$	$\hat{X}(\hat{t}) = \sum_{t=0}^{N-1} X(t)e^{\frac{2\pi i t \hat{t}}{N}}$
Linearity	
The Fourier transform is a linear operator.	The Fourier transform is a linear operator.
Orthogonality	
$\int_{-\infty}^{\infty} e^{2\pi i t(\hat{t}-\hat{t}')} dt = \delta(\hat{t}-\hat{t}')$	$\sum_{t=0}^{N-1} e^{\frac{2\pi i t(\hat{t}-\hat{t}')}{N}} = N\delta_N(\hat{t}-\hat{t}')$
where $\delta(\hat{t}-\hat{t}')$ is the Dirac delta function. That is,	where δ_N is the Kronecker delta function with its argument being considered modulo N . That is, $\delta_N(kN) = 1$, for integer k , otherwise $\delta_N = 0$.
$\int_b^a f(\hat{t})\delta(\hat{t})d\hat{t} = f(0)$ if 0 is in the interval (a,b) , otherwise	
$\int_a^b f(\hat{t})\delta(\hat{t})d\hat{t} = 0.$	
Inverse Transform	
If $\hat{X}(\hat{t}) = \int_{-\infty}^{\infty} X(t)e^{2\pi i t \hat{t}} dt$	If $\hat{X}(\hat{t}) = \sum_{t=0}^{N-1} X(t)e^{\frac{2\pi i t \hat{t}}{N}}$
then	then
$X(t) = \int_{-\infty}^{\infty} \hat{X}(\hat{t})e^{-2\pi i t \hat{t}} d\hat{t}$	$X(t) = \frac{1}{N} \sum_{\hat{t}=0}^{N-1} \hat{X}(\hat{t})e^{-\frac{2\pi i t \hat{t}}{N}}$
which we observe can be considered	which we observe can be considered
as $X(t) = \left\{ \int_{-\infty}^{\infty} \{\hat{X}(\hat{t})\}^* e^{2\pi i t \hat{t}} d\hat{t} \right\}^*$	as $X(t) = \frac{1}{N} \times \left\{ \sum_{\hat{t}=0}^{N-1} \{\hat{X}(\hat{t})\}^* e^{\frac{2\pi i t \hat{t}}{N}} \right\}^*$
or the complex conjugate of the Fourier transform of the complex conjugate of X .	or the complex conjugate of the Fourier transform of the complex conjugate of X , divided by N .
Convolution Theorem	
$\int_{-\infty}^{\infty} X(\tau)Y(t-\tau) d\tau = \int_{-\infty}^{\infty} e^{-2\pi i t \hat{t}} \hat{X}(\hat{t})\hat{Y}(\hat{t})d\hat{t}$	$\sum_{\tau=0}^{N-1} X(\tau)Y(t-\tau) = \frac{1}{N} \times \sum_{\hat{t}=0}^{N-1} e^{-\frac{2\pi i t \hat{t}}{N}} \hat{X}(\hat{t})\hat{Y}(\hat{t})$

TABLE 1—(Continued)

Usual	Finite Discrete
that is, the inverse Fourier transform of the product of the Fourier transforms.	that is, the inverse Fourier transform of the product of the Fourier transforms. NOTE: The convolution here must be considered as cyclic, i.e., the indices of X and Y must be interpreted modulo N .
Operational Calculus	
An operational calculus can be defined, based on the property that	An operational calculus can be defined, based on the property that
$\int_{-\infty}^{\infty} \left(\frac{\partial}{\partial t} X(t) \right) e^{2\pi i t \hat{t}} dt = -2\pi i \hat{t} \hat{X}(\hat{t})$	$\sum_{t=0}^{N-1} (\Delta X(t)) e^{\frac{2\pi i t \hat{t}}{N}} = \left(e^{-\frac{2\pi i \hat{t}}{N}} - 1 \right) \hat{X}(\hat{t})$
i.e., the Fourier transform of the derivative of a function is the Fourier transform of the function, multiplied by $-2\pi i \hat{t}$.	i.e., the Fourier transform of the (forward) difference of a function is the Fourier transform of the function, multiplied by $\left(e^{-\frac{2\pi i \hat{t}}{N}} - 1 \right)$.
	NOTE: The difference here must be considered cyclically, so that $\Delta X(t) = X(t+1) - X(t)$ becomes $\Delta X(N-1) = X(N) - X(N-1) = X(0) - X(N-1)$ for the case of $t = N - 1$.
Symmetries	
If X is real then \hat{X} is hermitian symmetric, i.e., $\hat{X}(\hat{t}) = \{\hat{X}(-\hat{t})\}^*$; if X is hermitian symmetric then \hat{X} is real.	If X is real then \hat{X} is hermitian symmetric, i.e., $\hat{X}(\hat{t}) = \{\hat{X}(N-\hat{t})\}^*$; if X is hermitian symmetric, then \hat{X} is real. If Y is imaginary, then \hat{Y} is hermitian antisymmetric, i.e., $\hat{Y}(\hat{t}) = -\{\hat{Y}(N-\hat{t})\}^*$; if Y is hermitian antisymmetric then \hat{Y} is imaginary.
If Y is imaginary, then \hat{Y} is hermitian antisymmetric, i.e., $\hat{Y}(\hat{t}) = -\{\hat{Y}(-\hat{t})\}^*$; if Y is hermitian antisymmetric then \hat{Y} is imaginary.	
	NOTE: The use of the terms hermitian symmetric and hermitian antisymmetric in the discrete case is consistent with that in the usual case if we interpret indices modulo N .
Shifting Theorems	
$\int_{-\infty}^{\infty} X(t+h)e^{2\pi i t \hat{t}} dt = e^{-2\pi i \hat{t} h} \hat{X}(\hat{t})$	$\sum_{t=0}^{N-1} X(t+h)e^{\frac{2\pi i t \hat{t}}{N}} = e^{-\frac{2\pi i \hat{t} h}{N}} \hat{X}(\hat{t})$
(These results are all readily proved from the definitions.)	

often the natural interpretation—as sampled data from a continuous infinite function.

Basic Algebra of Fast Fourier Transforms

At this point we will define the notation $e(X) = e^{2\pi i X}$ so that we may write expressions such as those of Table I in a simpler form. The two most important properties of $e(X)$ are that

$$e(X+Y) = e(X)e(Y)$$

and

$$e(X) = 1 \text{ if } X \text{ is an integer.}$$

Using this notation, the finite discrete Fourier transform of the sequence $X(t)$ is

$$\hat{X}(\hat{t}) = \sum_{t=0}^{N-1} X(t)e\left(\frac{t\hat{t}}{N}\right)$$

Suppose N has factors A and B so that $N = AB$.

Then writing $\hat{t} = \hat{a} + \hat{b}A$ and $t = b + aB$ where $a, \hat{a} = 0, 1, \dots, A-1$ and $b, \hat{b} = 0, 1, \dots, B-1$; we have

$$\begin{aligned} \hat{X}(\hat{a} + \hat{b}A) &= \\ &= \sum_{b=0}^{B-1} \sum_{a=0}^{A-1} X(b+aB)e\left(\frac{\{\hat{a} + \hat{b}A\}\{b+aB\}}{AB}\right) \\ &= \sum_{b=0}^{B-1} \sum_{a=0}^{A-1} X(b+aB)e\left(\frac{\hat{a}b}{AB} + \frac{\hat{a}a}{A} + \frac{\hat{b}b}{B} + \hat{a}\hat{b}\right) \\ &= \sum_{b=0}^{B-1} \sum_{a=0}^{A-1} X(b+aB)e\left(\frac{\hat{a}b}{AB}\right)e\left(\frac{a\hat{a}}{A}\right)e\left(\frac{\hat{b}b}{B}\right) \\ &\quad \text{as } \hat{a}\hat{b} \text{ is integral implies } e(\hat{a}\hat{b}) = 1 \\ &= \sum_{b=0}^{B-1} e\left(\frac{\hat{b}b}{B}\right)\left\{e\left(\frac{\hat{a}b}{AB}\right)\sum_{a=0}^{A-1} X(b+aB)e\left(\frac{a\hat{a}}{A}\right)\right\} \end{aligned}$$

If we define the B different sequences

$$W_b(a) = X(b+aB) \quad a = 0, 1, \dots, A-1$$

and the A different sequences

$$\begin{aligned} Z_{\hat{a}}(b) &= e\left(\frac{\hat{a}b}{AB}\right)\sum_{a=0}^{A-1} X(b+aB)e\left(\frac{a\hat{a}}{A}\right) \\ b &= 0, 1, \dots, B-1 \end{aligned}$$

we can write the above equation as

$$\hat{X}(\hat{a} + \hat{b}B) = \sum_{b=0}^{B-1} e\left(\frac{\hat{b}b}{B}\right) Z_{\hat{a}}(b)$$

where

$$Z_{\hat{a}}(b) = e\left(\frac{\hat{a}b}{AB}\right)\left\{\sum_{a=0}^{A-1} e\left(\frac{a\hat{a}}{A}\right)W_b(a)\right\}$$

We recognize

$$\sum_{b=0}^{B-1} e\left(\frac{\hat{b}b}{B}\right)Z_{\hat{a}}(b) \text{ and } \sum_{a=0}^{A-1} e\left(\frac{a\hat{a}}{A}\right)W_b(a)$$

as Fourier transforms themselves, applied to shorter sequences. Obtaining a subsequence by starting at the b th element and taking every B th element thereafter, in the manner $W_b(a)$ is obtained from $X(b+aB)$, is called decimating by B . Observe that the sequence $Z_{\hat{a}}(b)$ are not quite the sequence of frequency \hat{a} values from the transforms of these decimated sequences, but these values multiplied by a “twiddle factor” $e\left(\frac{\hat{a}b}{AB}\right)$.

The Fourier transform of the complete AB point sequence may thus be accomplished by doing the B different A point Fourier transforms, multiplying through by the appropriate twiddle factors, then doing the A different B point Fourier transforms. This is a recursive formula which defines the larger Fourier transform in terms of smaller ones. The total number of operations is now proportional to $AB(A+B)$ rather than $(AB)^2$ as it would be for a direct implementation of the definition, hence the name “Fast Fourier Transform”.

Associated observations:

1. Attention is drawn to the special advantage of factors 2 or 4, in that since a Fourier transform of a 2 or 4 point sequence may be done using only additions and subtractions, one stage of complex arithmetic may be avoided.
2. Although the recursive algorithm above may be implemented directly, it is much better to simulate the recursion as described in the following sections, since one may avoid the unnecessary calculation of some complex exponentials. With our programs, for example, the simulating the recursion takes only 2/5 as long for a 2^{10} point transform.*

* All programs discussed in this paper were implemented using ALCOR Algol 60 on the IBM 7094 at Princeton University.

3. The multi-dimensional finite discrete Fourier transform

$$\hat{X}(\hat{t}_1, \dots, \hat{t}_k) = \sum_{t_1=0}^{N_1-1} \cdots \sum_{t_k=0}^{N_k-1} X(t_1, \dots, t_k) e\left(\frac{t_1 \hat{t}_1}{N_1}\right) \cdots e\left(\frac{t_k \hat{t}_k}{N_k}\right)$$

can be computed efficiently by factorizing in each dimension separately, as above.

4. The algebra presented here is not quite that appearing in "An Algorithm for the Machine Calculations of Complex Fourier Series," by J. W. Cooley and J. W. Tukey.² The difference can be seen if we consider N as having three factors, $N = ABC$.

The Cooley-Tukey algebra then is

$$\begin{aligned} \hat{X}(\hat{c} + \hat{b}C + \hat{a}BC) &= \sum_{a=0}^{A-1} \sum_{b=0}^{B-1} \sum_{c=0}^{C-1} X(a + bA + cAB) e\left(\frac{(\hat{c} + \hat{b}C + \hat{a}BC) \cdot (a + bA + cAB)}{ABC}\right) \\ &= \sum_{a=0}^{A-1} \sum_{b=0}^{B-1} \sum_{c=0}^{C-1} X(a + bA + cAB) e\left(\frac{a(\hat{c} + \hat{b}C + \hat{a}BC)}{ABC}\right) \\ &\quad \cdot e\left(\frac{b(\hat{c} + \hat{b}C + \hat{a}BC)}{BC}\right) e\left(\frac{c(\hat{c} + \hat{b}C + \hat{a}BC)}{C}\right) \\ &= \sum_{a=0}^{A-1} e\left(\frac{a(\hat{c} + \hat{b}C + \hat{a}BC)}{ABC}\right) \sum_{b=0}^{B-1} e\left(\frac{b(\hat{c} + \hat{b}C)}{BC}\right) \\ &\quad \sum_{c=0}^{C-1} e\left(\frac{c\hat{c}}{C}\right) X(a + bA + cAB) \end{aligned}$$

If, rather than collecting on the unhatted variables as above, we choose to collect on the hatted variables, we obtain

$$\begin{aligned} \hat{X}(\hat{c} + \hat{b}C + \hat{a}BC) &= \sum_a^{A-1} \sum_b^{B-1} \sum_c^{C-1} X(a + bA + cAB) e\left(\frac{\hat{a}(a + bA + cAB)}{A}\right) \\ &\quad \cdot e\left(\frac{\hat{b}(a + bA + cAB)}{AB}\right) e\left(\frac{\hat{c}(a + bA + cAB)}{ABC}\right) \end{aligned}$$

$$= \sum_{a=0}^{A-1} e\left(\frac{\hat{a}a}{A}\right) \sum_{b=0}^{B-1} e\left(\frac{\hat{b}(a + bA)}{AB}\right) \sum_{c=0}^{C-1} e\left(\frac{\hat{c}(a + bA + cAB)}{ABC}\right) X(a + bA + cAB)$$

In both cases we may factor the twiddle factor outside the summation. If we do this we obtain in the first case

$$\begin{aligned} \hat{X}(\hat{c} + \hat{b}C + \hat{a}BC) &= \sum_{a=0}^{A-1} e\left(\frac{a\hat{a}}{A}\right) e\left(\frac{a[\hat{c} + \hat{b}C]}{ABC}\right) \\ &\quad \sum_{b=0}^{B-1} e\left(\frac{b\hat{b}}{B}\right) e\left(\frac{b\hat{c}}{BC}\right) \sum_{c=0}^{C-1} e\left(\frac{c\hat{c}}{C}\right) X(a + bA + cAB) \end{aligned}$$

(Cooley version)

In the second case we obtain

$$\begin{aligned} \hat{X}(\hat{c} + \hat{b}C + \hat{a}BC) &= \sum_{a=0}^{A-1} e\left(\frac{a\hat{a}}{A}\right) e\left(\frac{a\hat{b}}{AB}\right) \sum_{b=0}^{B-1} e\left(\frac{b\hat{b}}{B}\right) e\left(\frac{\hat{c}(a + bA)}{ABC}\right) \sum_{c=0}^{C-1} e\left(\frac{c\hat{c}}{C}\right) X(a + bA + cAB) \end{aligned}$$

(Sande version)

The difference between these two versions becomes important when we come to the details of implementing the fast Fourier transform. We go on to this next.

Fast Fourier Transforms Using Scratch Storage

Let us consider the Sande version applied to an X sequence stored in serial order. The summation over c represents a C point Fourier transform of points spaced AB apart. There is one such transform for each of the AB values of $a + bA$. The result of one such transform is a frequency sequence indexed by \hat{c} .

If we look at the twiddle factor $e\left(\frac{\hat{c}(a + bA)}{ABC}\right)$ we see that it depends upon \hat{c} and $a + bA$ in a very convenient manner. To introduce some descriptive terminology we may call \hat{c} the frequency for this analysis and $a + bA$ the displacement for this analysis. At this stage there are no free indices corresponding to replications. When we store the intermediate results in the scratch storage area, placing each of the C point Fourier transforms in contiguous blocks indexed by the displacement leads to elements stored at $\hat{c} + C(a + bA)$. The intermediate summation over b represents a B point Fourier transform with points spaced AC apart; one such transform for each of the AC values of $\hat{c} + aC$. The result of any

one of these transforms is a frequency sequence indexed by \hat{b} . The twiddle factor $e\left(\frac{a\hat{b}}{AB}\right)$ depends only upon a and \hat{b} , leaving \hat{c} as a free index. Here we would call \hat{b} the frequency for the analysis, a the displacement for the analysis and \hat{c} the replication index. We would store the intermediate results at $\hat{c} + \hat{b}C + aBC$. In this case the contiguous blocks are spaced out by the replication factor C . The outer summation over a represents an A point Fourier transform with points spaced BC apart; one such transform for each of the BC values of $\hat{c} + \hat{b}C$. The result of any one of these transforms is a frequency sequence indexed by \hat{a} . There is no twiddle factor in this case. Here we would call \hat{a} the frequency and $\hat{c} + \hat{b}C$ the replication index. There is no displacement at this stage. We would store the results at $\hat{c} + \hat{b}C + \hat{a}BC$. At this point we see that the results are stored in serial order of frequency.

When we compute one of the transforms we may wish to include the twiddle factor in the coefficients for the values of the replication index before computing new coefficients for a different displacement. If the Fourier transform is on two points there appears to be no advantage to either choice. For four points it is more economical to do the Fourier transform and then multiply by the twiddle factor. In the other cases it is more efficient if the twiddle factor is absorbed into the transform coefficients.

If we were to use the Cooley version on a sequence stored in serial order we would obtain an algorithm which differs only in minor details. The summation over c represents a C point Fourier transform of points spaced AB apart. The twiddle factor $e\left(\frac{b\hat{c}}{BC}\right)$ depends upon the frequency \hat{c} and displacement b , leaving a free as a replication index. The elements are stored at $\hat{c} + aC + bAC$. The intermediate summation over b represents a B point Fourier transform of points spaced AC apart. The twiddle factor $e\left(\frac{a(\hat{c} + \hat{b}C)}{ABC}\right)$ depends upon the combined frequency $\hat{c} + \hat{b}C$ and the displacement a . There is no free index. The intermediate results are stored at $\hat{c} + \hat{b}C + aBC$. The outer summation represents an A point Fourier transform of results spaced BC apart. There is no twiddle factor in this case. The final results would be stored in serial order at $\hat{c} + \hat{b}C + \hat{a}BC$.

These two reductions are essentially identical from

an algorithmic viewpoint when we use storage in this manner. We have only reversed the roles of the hatted and unhatted variables in the formation of the twiddle factors.

To obtain the general case of more than three factors we proceed to group our factors in three groups, for example $N = (P_1 \dots P_{j-1})P_j(P_{j+1} \dots P_n)$. If we identify

$$\begin{aligned} A &= P_1 \dots P_{j-1} \\ B &= P_j \\ C &= P_{j+1} \dots P_n \end{aligned}$$

and perform the above reductions we find that after two steps with this identification we arrive at exactly the same place as after only one step with the identification

$$\begin{aligned} A &= P_1 \dots P_{j-2} \\ B &= P_{j-1} \\ C &= P_j \dots P_n \end{aligned}$$

We can think of this as moving factors from the "A" part through the "B" part to the "C" part.

When we write a program to implement this, we set up a triply nested iteration. The outer loop selects the current factor being moved and sets up the limits and indexing parameters for the inner loops. The intermediate loop, in which we would compute the twiddle factor, corresponds to the displacement. The inner loop provides indexing over the replication variables.

Fast Fourier Transforms in Place

Let us reconsider the storage requirements for our algorithm. In particular, let us consider one of the small Fourier transforms on c for some value of $a + bA$. It is only during this transform that we will need this set of intermediate results. We have just vacated the C cells $a + bA + cAB$ (indexed by c) and are looking about for C cells in which to store our answers (indexed by \hat{c}). Why not use the cells that have just been vacated?

Having made this observation, let us reexamine the algorithm corresponding to the Sande factorization. The summation on c represents a C point Fourier transform of points spaced AB apart. The twiddle factor $e\left(\frac{\hat{c}(a+bA)}{ABC}\right)$ depends upon the frequency \hat{c} and the displacement $a + bA$. There is no replication index. The intermediate results are stored at $a + bA + \hat{c}AB$. The summation on b represents a B point Fourier transform of points A apart. The

twiddle factor $e\left(\frac{a\hat{b}}{AB}\right)$ depends upon the frequency \hat{b} and the displacement a . The replication index is \hat{c} . We treat each of the blocks of length AB indexed by \hat{c} in an exactly equivalent manner. The summation over a represents contiguous A point Fourier transforms. The frequency is \hat{a} and the replication index is $\hat{b} + c\hat{B}$. There is no displacement. The final answers are then stored at $\hat{a} + \hat{b}A + \hat{c}AB$. The use of "displacement" and "replication" is now much more suggestive and natural than when we used these while describing the use of scratch storage.

The more general case of more than three factors is again obtained by grouping to obtain "A", "B", and "C" and then moving factors from the "A" part to the "C" part; the program being written as a triply nested iteration loop.

When we reexamine the Cooley factorization for serially stored data, we will see that it has an unpleasant feature. The summation over c represents a C point Fourier transformation of points spaced AB apart. The twiddle factor $e\left(\frac{b\hat{c}}{AB}\right)$ depends only upon b and \hat{c} , leaving a as a free index. The intermediate results are stored at $a + bA + \hat{c}AB$. The summation over b represents a B point Fourier transform of points spaced A apart. The twiddle factor $e\left(\frac{a(\hat{c} + \hat{b}c)}{ABC}\right)$ depends upon the displacement a and the combined frequency $\hat{c} + \hat{b}c$. For data which was originally stored serially, this is not convenient to compute. The intermediate results are stored at $a + \hat{b}A + \hat{c}AB$. The summation over a represents contiguous A point Fourier transforms. The final results are stored at $\hat{a} + \hat{b}A + \hat{c}AB$. More than three factors can be handled in the same manner as before.

The common unpleasant feature of both of these algorithms is that the Fourier coefficient for frequency $\hat{c} + \hat{b}c + \hat{a}BC$ is stored at $\hat{a} + \hat{b}A + \hat{c}AB$. Such a storage scheme may be described as storage with "digit reversed subscripts." Before we may use our Fourier coefficients we must unscramble them.* Unscrambling has been found to require only a small proportion of the total time for the algorithm. A very elegant solution corresponds to running two counters, one with normal digits (easily obtained by simply

*This problem has nothing to do with the representation of numbers in any particular machine, and unless the machine has a "reverse digit order" instruction (e.g., "reverse bit order" for a binary machine), no advantage can be taken of such representation.

TABLE II
TIME FOR A 1024 POINT TRANSFORM
BY VARIOUS METHODS

Method	Time
radix 2	2.0 seconds
radix 4 (also radix 4 + 2 and mixed radices)	1.1 seconds
radix 2 (recursively implemented)	5.2 seconds
Goertzel's method	59.1 seconds

incrementing by one) and one with reversed digits (easily obtained by nesting loops with the innermost stepping by the largest increments) and recopying from one array to another where one counter is used for each array. If all of the factors are the same, the required interchanges become pairwise and it is possible to do the unscrambling in place. One may unscramble the real and imaginary parts separately, so scratch space could be generated by backing one or the other onto auxiliary store. A slower method of unscrambling is to follow the permutation cycles so that no scratch storage is needed.

Two implementations, both for serially stored sequences, have been described. Alternately, two implementations, both for data stored with digit reversed subscripts, may be developed: in this instance the Cooley factorization has more convenient twiddle factors. For the last two, the final results are correctly stored—the "unscrambling" having been done first. Digit reversed subscripts may arise because we have

1. scrambled serially stored data,
2. not unscrambled after a previous Fourier transform,
3. generated data in scrambled order.

We have written various Fourier transform programs in the manner described above. These include one- and multi-dimensional versions of radix 2 (all factors equal to 2), radix 4 (all factors equal to 4), radix 4 + 2 (all factors equal to 4 except perhaps the last, which may be 2), and mixed radices (N is factored into as many 4's as possible, a 2 if necessary, then any 3's, 5's or other primes). Times required to transform a 1024 point sequence are given in Table II, including the time required by a recursive radix 2 transform and by the pre-fast Fourier transform "efficient" Goertzel's method³ (which still requires order N^2 operations) for comparison. Our standard tool is the radix 4 + 2 Fourier transform which combines the speed of radix 4 with the

flexibility of radix 2. The mixed radices Fourier transform is used when other factors, for example 10, are desired. The mixed radix is used less as it is more bulky (requiring scratch storage for unscrambling as well as being a larger program) and is less thoroughly optimized than the radix $4 + 2$ Fourier transform.

Fast Fourier Transforms Using Hierarchical Store

As we become interested in doing larger and larger Fourier transforms, we reach a point where we may not be able to have all of the data in core simultaneously, and some must be kept in slower store such as drum, disk, or tape. At the other end of the scale, small amounts of very fast memory will be available with some of the "new generation" computers—memory much faster than the rest of core. Automatic systems for providing "virtual core" obscure the distinctions within this hierarchy of memories, but at great loss of efficiency for Fourier transforms. On the other hand, we can apply the basic recursion formula of the section *Basic Algebra of Fast Fourier Transforms* to employ hierarchical store in such a way as to operate on sequences large enough to require the slower store, yet to run little slower than if all the memory had the speed of the faster store.

In particular, when we factor N into the two factors A and B , we choose A as the largest Fourier transform that can be done in the faster store. We then compute the transforms (including unscrambling) of the B subsequences obtained by decimating the original sequence by B . We next multiply through by the appropriate twiddle factors, and do the B point transforms. Thus, other than decimating and recopying, which amounts to a matrix transpose, all the computations are done in the faster store, the loading and unloading of which can be overlapped. As an example, consider using a 32K machine with disk to Fourier transform a quarter million (2^{18}) point sequence (point here means complex number) initially on tape, returning the transform to tape. We will assume that we are willing to do transforms as large as 4096 points (2^{12}) in core.

The first step is to create 64 (2^6) files on disk, each of length 4096. The tape is read into core and the various decimated sequences read out onto disk—the zeroth point going into the zeroth file, the first into the first, etc. up to the sixty-third into the sixty-third, then the sixty-fourth point starting back in the zeroth file again, etc. Appropriately buffered, this can run almost at disk transfer speed.

The 64 files are now brought into core one at a time. When a file comes in, it is Fourier transformed using a standard in-core transform, then the points are multiplied by the twiddle factor $e(\hat{a}b/2^{18})$, where b is the number of the file (0 through 63) and \hat{a} is the frequency index for the particular point. The file is then written out again. By choosing to use transforms of only 4096 points, we can fully buffer these operations, one file being Fourier transformed while the previous file is being written out and the succeeding file read in.

We now start on the second set of transforms. Here we make use of the random access feature of disk to read simultaneously from each of the 64 files, since there are 4096 subsequences of length 64, obtained by taking one element from each file. Each subsequence is brought in, transformed, and then returned to its previous position on the disk, that is, as one number in each of the files 0 through 63, according to the frequency index. Again we may completely buffer this operation so that the disk time comes free.

Finally we output our Fourier transform onto tape, sequentially writing out each of the files, 0 through 63.

Rough calculations for the IBM 7094 Mod I show that the initial decimating is limited by disk speed, and takes about 3 minutes, that the first 64 (4096 point) transforms will take a total of just under 6 minutes and are compute-bound so that disk time is free, that the last 4096 (64 point) transforms will take a total of about 3 minutes, again compute-bound so that disk time is free, and the final output is completely limited by disk speed, and takes about 3 minutes. Thus our quarter-million point Fourier transform has cost us only about 15 minutes, 9 minutes of which is computing, the same 9 minutes which would be required if the machine had enough core to do the entire problem internally, and 6 minutes of which is strictly tape to disk or disk to tape transmission that could be avoided if the sequence was initially available, and the final answers acceptable, in appropriate form or could at least be overlapped with other computing.

Roundoff Considerations

So far we have discussed the fast Fourier transform as though it could be done with complete accuracy. What happens when the algorithm is carried out in a real fixed word-length computer using floating point arithmetic? How does the error in doing

this compare with the error from a direct application of the definition?

We may approach this question two ways—by strict bounds or by empirical values determined by experimentation. We shall derive bounds for the ratio of the Euclidean norm* of the error to the Euclidean norm of the data sequence, showing that whereas for direct application of the defining formula we can only bound this ratio by $1.06\sqrt{N} (2N)^{3/2}2^{-b}$, if we factor N into $n_1n_2\dots n_k$ and use the fast Fourier transform we can bound the ratio by $1.06\sqrt{N} \left\{ \sum_j (2n_j)^{3/2} \right\} 2^{-b}$. In particular, if all the n_j are the same, so that $N = n^k$, then the ratio is less than $1.06\sqrt{N} k(2n)^{3/2}2^{-b}$, which is $k/n^{3/2(k-1)}$ times that for the direct calculation. (b here is the number of bits used in the mantissa of the floating-point representation). We shall then see from empirical results that the form of the bound is actually a fairly good description of the form of the observed errors, although the numerical constant is somewhat larger than is typical of actual trials.

Theorem: If we use the defining formula to Fourier transform a sequence $X(t)$ of length N in a machine using floating point arithmetic and a fixed word length with a b bit mantissa, then

$$\|f\ell(\hat{X}) - \hat{X}\|_E < 1.06 \sqrt{N} (2N)^{3/2}2^{-b} \|\hat{X}\|_E$$

Proof: We require the following lemma from Wilkinson⁹ (p. 83):

Lemma: If A is a real p by q matrix and B is a real q by r matrix and if $f(\cdot)$ denotes the result of floating point computation then

$$\|f\ell(AB) - AB\|_E < 1.06q2^{-b} \|A\|_E \|B\|_E$$

*The Euclidean norm of a vector or sequence is the square root of the sum of the squares of the elements.

$$\begin{aligned} \|f\ell(\hat{X}) - \hat{X}\|_E &= \sqrt{N} \|f\ell M_k f\ell M_{k-1} \dots f\ell M_1 X - M_k M_{k-1} \dots M_1 X\|_E \\ &= \sqrt{N} \|f\ell M_k f\ell M_{k-1} \dots f\ell M_1 X - M_k f\ell M_{k-1} f\ell M_{k-2} \dots f\ell M_1 X \\ &\quad + M_k f\ell M_{k-1} \dots f\ell M_1 X - M_k M_{k-1} f\ell M_{k-2} \dots f\ell M_1 X \\ &\quad \dots + M_k M_{k-1} \dots f\ell M_1 X - M_k M_{k-1} \dots M_1 X\|_E \\ &\leq \sum_{j=1}^k \sqrt{N} \|M_k \dots M_{j+1} (f\ell M_j f\ell M_{j-1} \dots f\ell M_1 X - M_j f\ell M_{j-1} \dots f\ell M_1 X)\|_E \end{aligned}$$

but since the M_j are orthogonal

$$= \sqrt{N} \sum_{j=1}^k \|f\ell M_j (f\ell M_{j-1} \dots f\ell M_1 X) - M_j (f\ell M_{j-1} \dots f\ell M_1 X)\|_E$$

q is the extent of the summation in the matrix multiplication.

The definition of the Fourier transform of $X(t)$ is also the definition of the multiplication of the vector $X(t)$ by the matrix $\{e(i\hat{t}/N)\}_{\hat{t}t}$.

Expressing this in real arithmetic, we multiply the real sequence $\{Re(X(t)), Im(X(t))\}$ by the real matrix

$$\begin{pmatrix} C & -S \\ S & C \end{pmatrix}$$

where $C = \{\cos 2\pi\hat{t}/N\}_{\hat{t}t}$ and $S = \{\sin 2\pi\hat{t}/N\}_{\hat{t}t}$. The norm of this $2N \times 2N$ real matrix is

$$\sum_{t=0}^{N-1} \sum_{\hat{t}=0}^{N-1} \{\cos^2 2\pi\hat{t}t/N + \sin^2 2\pi\hat{t}t/N + \cos^2 2\pi\hat{t}t/N + \sin^2 2\pi\hat{t}t/N\} = \sqrt{2N^2}$$

Since the extent of the summation is $2N$, we have by the lemma that $\|f\ell(\hat{X}) - \hat{X}\|_E < 1.06 \times 2N \sqrt{2N^2} 2^{-b} \|\hat{X}\|_E$.

The proof is completed by observing that $\|\hat{X}\|_E = \sqrt{N} \|X\|_E$.

Theorem: If we use the fast Fourier transform, factoring N into $n_1n_2\dots n_k$, to transform a sequence $X(t)$ in a machine using floating point arithmetic and a fixed word length with a b bit mantissa, then,

$$\|f\ell(\hat{X}) - \hat{X}\|_E < 1.06 \sqrt{N} \sum_{j=1}^k (2n_j)^{3/2}2^{-b} \|\hat{X}\|_E$$

Proof: Here we represent the transform as a sequence of matrix multiplies (one for each factor), and employ the above lemma. Specifically, the fast Fourier transform is equivalent to evaluating $\sqrt{N} M_k M_{k-1} \dots M_1 X$ where M_j is a matrix consisting of N/n_j disjoint n_j point Fourier transforms (including the twiddle factors for the next stage), rescaled to be orthogonal. Thus

When we compute the bound for $\|f\mathcal{L}(M_j Y) - M_j Y\|_E$ we find that since M_j may be partitioned into N/n_j disjoint blocks, we can bound the error from each block separately, getting, since in real arithmetic each block is $2n_j$ square, and the norm of the block is $\sqrt{2n_j}$, a bound of $1.06(2n_j)^{3/2}2^{-b}\|Y_S\|_E$ where Y_S is the appropriate part of Y . By using Pythagoras' theorem this yields

$$\|f\mathcal{L}(M_j Y) - M_j Y\|_E < 1.06(2n_j)^{3/2}2^{-b}\|Y\|_E$$

Finally we observe that except for error of order $N2^{-b}\|X\|_E$

$$\|f\mathcal{L}M_{j-1}f\mathcal{L}M_{j-2}\dots f\mathcal{L}M_1X\|_E = \|M_j\dots M_1X\|_E$$

Immediately we have $\|M_j\dots M_1X\|_E = \|X\|_E$ because of the orthogonality, so

$$\|f\mathcal{L}(\hat{X}) - \hat{X}\|_E < 1.06\sqrt{N}\sum_{j=1}^k(2n_j)^{3/2}2^{-b}\|\hat{X}\|_E$$

Corollary: If a sequence is transformed and then the inverse transform applied to the result, the norm of the difference between the initial and final sequences can be bounded by $2 \times 1.06(2N)^{3/2}2^{-b}\|X\|_E$ if we use the definition, or $2 \times 1.06\sum_j(2n_j)^{3/2}2^{-b}\|X\|_E$ if we use the fast Fourier transform.

With this corollary in mind, an experiment we can readily do is to take a sequence, transform, inverse transform, and then compute the norm of the difference between the original and resultant sequences, dividing this by the norm of the original sequence. Table III gives these ratios for random Gaussian sequences of length 2^k , $k = 1, \dots, 12$, when the transforms were computed using: (1) a radix 2 transform; (2) a radix 4 + 2 transform; (3) a radix 4 + 2 transform with rounding instead of truncation; (4) a Goertzel's method transform; and (5) a transform which directly implemented the definition. The results of three replications of the experiment are given. Figure 1 is a plot of the average ratio divided by $\log_2 N$ against $\log_2 N$ to show the essentially linear dependence (for the factored transforms) of the ratio on $\log_2 N$. Attention is drawn to the common scaling factor of 10^{-8} and the consistency of the tabled ratios for the larger transforms.

USING FAST FOURIER TRANSFORMS

Classical Usage with Partial Differential Equations

One of the classical motivations for the study of Fourier transforms has been the application of find-

TABLE III
OBSERVED RATIOS OF ERROR NORM TO SEQUENCE NORM FOR THREE RANDOM SEQUENCES AT EACH LENGTH.
(in units of 10^{-8})

Log ₂ N	Radix 4+2				
	Radix 2	Radix 4+2	with rounding	Goertzel's method	Defining formula
1	1.03	(1.03)	.46	2.60	4.27
	.53	(.53)	.13	3.91	1.54
	.57	(.57)	.00	2.73	4.14
2	3.43	1.09	.92	9.73	5.15
	2.23	.92	1.05	12.6	3.31
	1.89	1.96	1.07	15.8	4.14
3	4.04	4.99	2.65	32.8	9.75
	5.57	5.58	2.69	16.6	13.0
	4.51	5.01	2.39	34.4	7.83
4	9.14	7.11	2.12	98.9	17.7
	8.64	5.92	2.91	91.5	18.4
	7.76	6.62	2.71	121.	17.1
5	10.7	11.0	4.58	202.	33.8
	12.7	12.2	4.44	258.	36.7
	11.9	11.4	5.40	198.	36.2
6	13.2	11.2	3.38	548.	69.1
	14.4	12.0	3.70	787.	75.2
	14.0	10.2	3.59	806.	63.8
7	17.6	17.0	6.24	1290.	143.
	16.9	16.3	6.74	1990.	141.
	17.3	16.8	6.78	1900.	135.
8	20.0	16.3	4.82	7050.	286.
	20.1	16.6	5.09	3490.	288.
	20.7	16.2	4.66	5090.	269.
9	22.8	21.6	7.61	13700.	579.
	22.8	21.4	7.81	10700.	578.
	22.9	21.7	7.91	11500.	561.
10	25.2	21.0	5.82	32100.	1170.
	25.6	21.0	5.44	27600.	1160.
	25.8	21.0	5.44	29900.	1140.
11	28.1	26.2	8.70	—	—
	28.1	26.1	8.56	—	—
	28.5	26.5	8.53	—	—
12	30.7	25.5	6.30	—	—
	30.7	25.5	6.21	—	—
	31.1	25.7	6.21	—	—

ing solutions of partial differential equations. Hockney⁴ gives a discussion of numerical solutions of partial differential equations. He considers the problem

$$\frac{\partial^2 \Phi(x,y)}{\partial x^2} + \frac{\partial^2 \Phi(x,y)}{\partial y^2} = \rho(x,y)$$

$0 \leq x \leq \ell$
 $0 \leq y \leq m$

with boundary conditions $\Phi(x,y) = 0$ if $x = 0, \ell$ or $y = 0, m$. The solution is obtained by discretizing

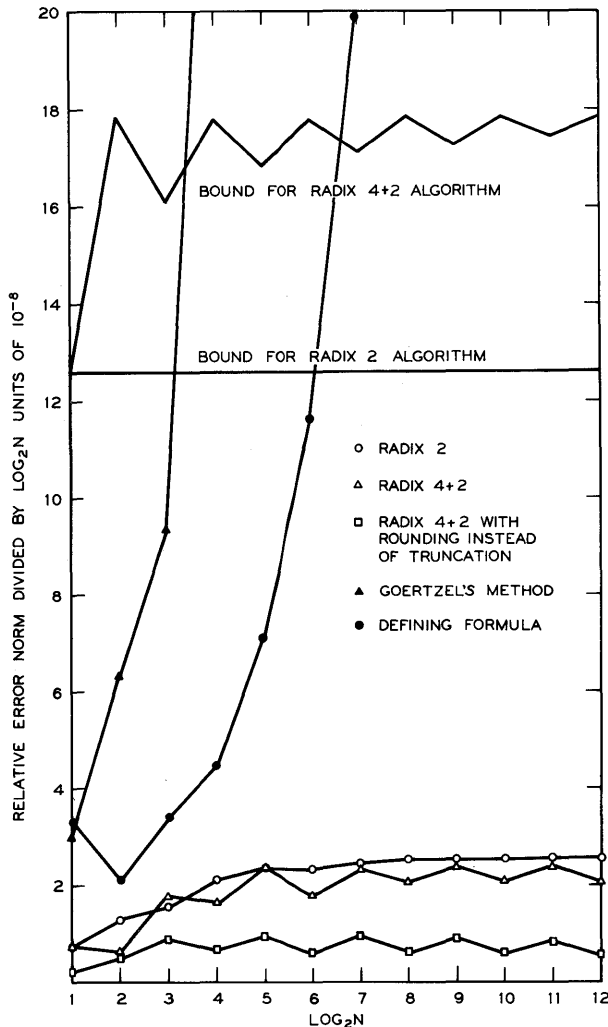


Figure 1. Observed values of relative error norm/Log₂N.

onto an $n \times n$ grid (Hockney considers $n = 48$, one case of $n = 12 \times 2^q$, $q = 0, 1, 2, \dots$). A Fourier transform with respect to x reduces the problem to the solution of a system of equations for the y coordinate. By using a 12 point formula from Whittaker and Robinson,⁸ Hockney is able to achieve a Fourier transform in $n^2/36$ operations.* If rather he were to use a fast Fourier transform for $n = 2^p$, $p = 0, 1, 2, \dots$ he could achieve a Fourier transform in approximately $\frac{3}{2} n \log_2 n$ operations. This estimate

of operations leads to a net count of $6n^2 \log_2 n$ for Hockney's method. Under this accounting, Hockney's method is much superior to the other methods he considers. It must be cautioned that the factoring

*Operation in this section means a real operation of a multiplication and an addition.

of 12×2^2 which Hockney uses for his Fourier transform will be equivalent to the fast Fourier transform. The apparent reduction only comes into play as the size, n , is increased past 48. As Hockney points out, the calculation of Fourier coefficients has long been recommended (and avoided because of cost) as a method of solving partial differential equations. Perhaps the fast Fourier transform will render the solution of partial differential equations both more economical and more straightforward.

Least Squares Approximation by Trigonometric Polynomials

Often one may want to approximate a given sequence by a band-limited sequence, that is, a sequence generated by a low order trigonometric polynomial. A possible example could be the determination of filter coefficients to approximate a desired transfer function.⁵ The use of expansions in terms of orthogonal functions to obtain least square approximations is well known. The Fourier transform of a sequence gives its expansion in terms of the mutually orthogonal complex exponentials, hence to obtain the coefficients for the approximation, we simply retain the low order coefficients from the Fourier transform. We may compare the approximation to the original by looking at the inverse Fourier transform of the coefficients after replacing the unwanted coefficients by zeros.

If we had wanted a positive definite approximation, we could have approximated the square root of the original sequence. The square of the approximation to the root is then the positive approximation. Squaring a sequence like this has the effect of convolving the Fourier coefficients. Other variations in the approximating problem may be handled by similar specialized techniques.

A caution: Gibbs phenomenon is a persistent problem when using least squares, particularly in the case of trigonometric approximation. In some situations, the problem may be sufficiently bad to warrant the use of other approximation criteria.

Numerical Convolutions

To date, the most important uses of the fast Fourier transform have been in connection with the convolution theorem of Table I. Some uses of numerical convolutions are the following:

Auto- and Cross-Covariances: 1. METHOD AND TIMING CONSIDERATIONS. In the analysis of time

series by digital spectrum analysis, there are many computational advantages to the “indirect” method of spectral estimation by computing the autocovariance and then Fourier transforming rather than the “direct” method by obtaining the periodogram and then smoothing.¹ For example, it is more practical to consider examining a spectrum through several different spectral windows by the indirect method. Similarly, when analyzing a pair of series with co- and quadrature-spectra, we often find the cross-covariance of the two series computationally convenient.

The first major use of the convolution theorem, by Sande,⁶ was to compute the auto-covariance

$$R_{XX}(\tau) = \frac{1}{N} \sum \{X(t)\} * X(t+\tau) \quad \tau = 0, \pm 1, \pm 2, \dots, \pm L$$

and cross-covariance

$$R_{XY}(\tau) = \frac{1}{N} \sum \{X(t)\} * Y(t+\tau) \quad \tau = 0, \pm 1, \pm 2, \dots, \pm L.$$

where the summations are overall values of t for which the products are defined. (These are not in the form of Table I but can be readily put into that form.) The convolution theorem may lead to improvements by a factor of 20 in computing time as compared to the summing of lagged products. The two major problems are that these are not defined cyclically and that N may not be convenient for use of the fast Fourier transform. Appending zeros to one or both ends of the series solves both problems.

To see this, extend X to length $N' \geq N + L$ by adding zeros so that

$$\begin{aligned} X'(t) &= X(t) & t &= 0, 1, \dots, N-1 \\ X'(t) &= 0 & t &= N, \dots, N'-1 \end{aligned}$$

Extend the Y series similarly. Fourier transform both new sequences to obtain

$$\begin{aligned} \hat{X}'(\hat{t}) &= \sum_{t=0}^{N'-1} X'(t) e\left(\frac{t\hat{t}}{N'}\right) \\ \hat{Y}'(\hat{s}) &= \sum_{s=0}^{N'-1} Y'(s) e\left(\frac{s\hat{s}}{N'}\right) \end{aligned}$$

Inverse transform the sequence formed by the product of the Y' transform and the complex conjugate of the X' transform.

$$C(\tau) = \frac{1}{N'} \sum_{\hat{t}=0}^{N'-1} \{\hat{X}'(\hat{t})\} * \hat{Y}'(\hat{t}) e\left(-\frac{\hat{t}\tau}{N'}\right)$$

$$\begin{aligned} &= \frac{1}{N'} \sum_{\hat{t}=0}^{N'-1} \sum_{t=0}^{N'-1} \sum_{s=0}^{N'-1} \\ &\quad \{X'(t)\} * Y'(s) e\left(-\frac{\hat{t}\tau}{N'}\right) e\left(-\frac{t\hat{t}}{N'}\right) e\left(\frac{s\hat{s}}{N'}\right) \\ &= \frac{1}{N'} \sum_{t=0}^{N'-1} \sum_{s=0}^{N'-1} \\ &\quad \{X'(t)\} * Y'(s) \sum_{\hat{t}=0}^{N'-1} e\left(\frac{\hat{t}(s-t-\tau)}{N'}\right) \end{aligned}$$

which by the orthogonality condition of Table I gives

$$\begin{aligned} &= \sum_{t=0}^{N'-1} \sum_{s=0}^{N'-1} \{X'(t)\} * Y'(s) \delta_{N'}(s-t-\tau) \\ &= \sum_{t=0}^{N'-1} \{X'(t)\} * Y'(t+\tau) \end{aligned}$$

Recall that the indices must be interpreted modulo N' . For $\tau < N' - N$, this expression is just N times the cross-covariance R_{XY} because then the extraneous products are precisely zero.

Note that the auto-covariance is precisely the cross-covariance of a series with itself, so that the outline just given also describes how to compute the auto-covariance. On the other hand, we may take advantage of the fact that in most time series problems the series are real to compute two auto-covariances at one time. We use one series as the real part and the other series as the imaginary part in forming the sequence which we Fourier transform, using the symmetry properties from Table I to separate them later.

Exactly how much faster the convolution theorem approach is than the lagged products depends not only on the length of series and proportion of the available lags desired, but also on details of the programs used. There are $\frac{(2N-L)(L+1)}{2}$ multiplications and additions, plus associated indexing operations, in summing lagged products. On the other hand, the number of operations required for the Fourier transforms (approximately proportional to $N' \log N'$) depends on our choice of N' which can vary according to the availability of Fourier transform routines, a routine which can handle mixed factors being much more flexible than a radix $4 + 2$ routine which restricts N' to be a power of two. Figure 2 gives a comparison of direct summing and

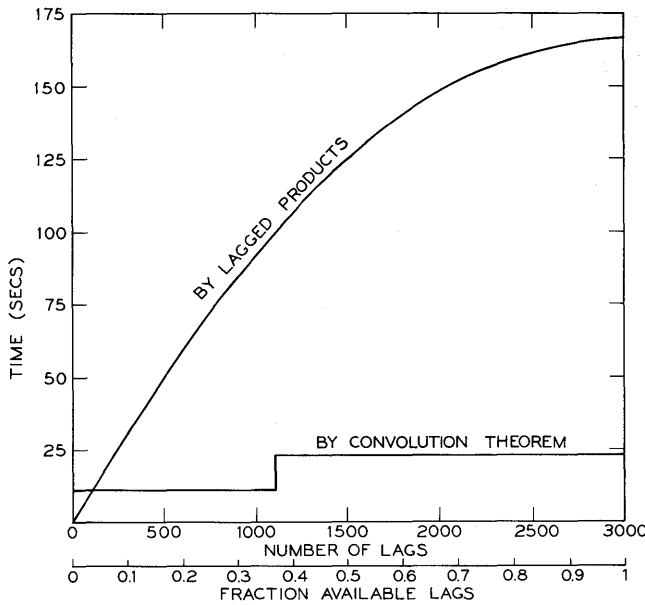


Figure 2. Computation times for auto-covariance of a 3000 point series, using convolution theorem or summing lagged products.

convolution theorem times using our radix 4 + 2 transform routine. Figure 3 shows (for this program) the regions in which either direct summing or the convolution theorem is more efficient as functions of N and L .

2. EXTRA-LONG SERIES. When we have to compute auto- or cross-covariances of extra-long series, we could just continue to use the method of the previous section, doing the oversize Fourier transforms by the method outlined in the section *Fast Fourier Transforms Using Hierarchical Store*. A simpler and occasionally more efficient method is also available. What we do instead is to partition the series X and Y into contiguous disjoint segments, and express the cross-covariance R_{XY} in terms of the cross-covariances between these various subseries.

A simple diagram will illustrate the method. In Fig. 4 we have plotted all the products of the elements of series Y against those of series X , the plotting position corresponding to the values of the respective indices. The cross-covariance $R_{XY}(\tau)$ then corresponds to $1/N$ times the sum over all elements on a line parallel to the main diagonal, but τ elements above it. Such a line is shown on the diagram.

As an example, we consider the problem of computing R_{XY} for $\tau = 0, \pm 1, \dots, \pm L$, when the largest cross-covariance we can compute by the method of the sub-section on Method and Timing Considera-

tions is for series of length $N/5$. We are thus interested in those products in the hatched area of the diagram. The segments described above are indicated by the dotted lines; they are labeled X_0, X_1, \dots, X_4 and Y_0, Y_1, \dots, Y_4 . We now see how $R_{XY}(\tau)$ may be calculated, by summing the appropriate cross-covariances of X_i with Y_j . For instance, for the particular value of τ illustrated,

$$R_{XY}(\tau) = \frac{1}{5} \left\{ R_{X_0Y_0}(\tau) + R_{X_1Y_0}\left(\tau - \frac{N}{5}\right) + R_{X_1Y_1}(\tau) + R_{X_2Y_1}\left(\tau - \frac{N}{5}\right) + R_{X_2Y_2}(\tau) + R_{X_3Y_2}\left(\tau - \frac{N}{5}\right) + R_{X_3Y_3}(\tau) + R_{X_4Y_3}\left(\tau - \frac{N}{5}\right) + R_{X_4Y_4}(\tau) \right\}$$

Filtering: DIGITAL FILTERS. Often, when we have a set of data which is indexed by displacement or time, we wish to “smooth” our data to reduce the effect of unwanted “noise”. If our original data were represented by $\{X(t)\}$ then forming

$$Y(t) = \frac{X(t) + X(t+1) + X(t+2)}{3}$$

would represent one method of smoothing our data. More generally, we could choose to use any arbitrary weights we pleased so that

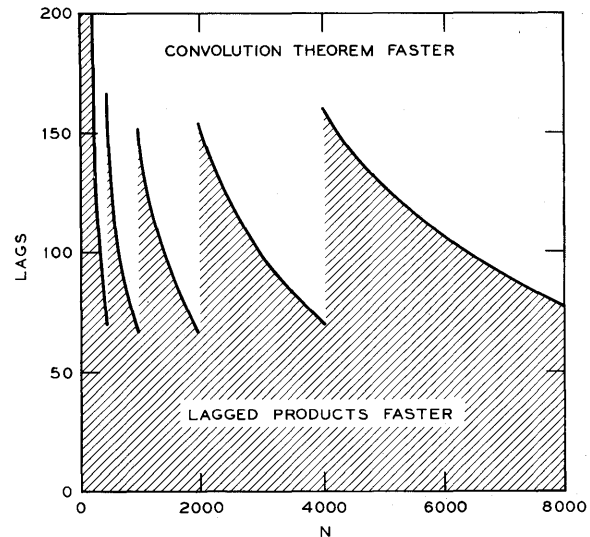


Figure 3. Region where using convolution theorem is faster than summing lagged products.

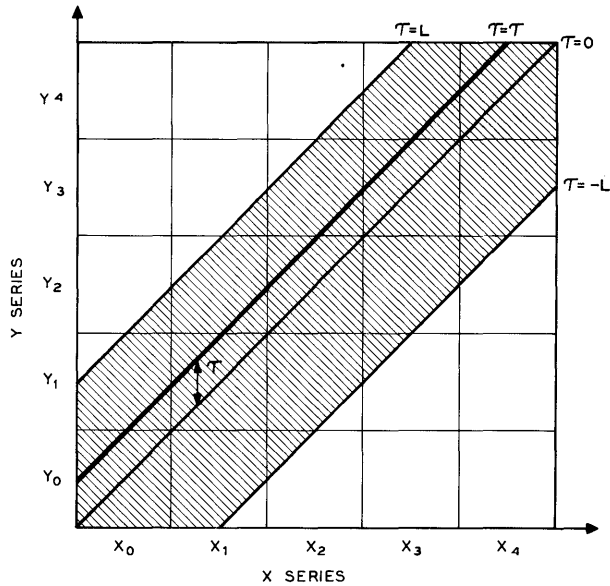


Figure 4. Diagram of cross products for computing cross-covariance.

$$Y(t) = c(k)X(t) + c(k-1)X(t+1) + \dots + c(o)X(t+k) = \sum_{j=0}^k c(k-j)X(t+j).$$

If, rather than smoothing, we had desired to estimate moving short-term integrals (possibly with a kernel) we would again be able to express the result as a summation over the data points weighted by suitable coefficients. The estimation of derivatives also leads to a similar expression.

Let us consider the application of such a moving average to a complex exponential of a fixed frequency. Thus our data would be $X(t) = e^{i\omega t} = e(ft)$ and the result of the moving average would be

$$\begin{aligned} Y(t) &= c(k)X(t) + \dots + c(o)X(t+k) \\ &= \sum_{j=0}^k c(k-j)X(t+j) \\ &= \sum_{j=0}^k c(k-j)e(f \times \{t+j\}) \\ &= e(ft) \sum_{j=0}^k c(k-j)e(fj). \end{aligned}$$

We can thus represent $Y(t)$ as the product of $e(ft)$ and $A(f) = \sum_{j=0}^k c(k-j)e(fj)$. $A(f)$ is called the

(complex) gain of the moving average (or filter) determined by the coefficients $c(o), \dots, c(k)$. Moving averages are linear in the data, for example, the moving average of $U(t) + V(t)$ is the same as the sum of the separate moving averages of $U(t)$ and $V(t)$. Thus if we consider our data as being composed of many Fourier components, we see that the moving average affects each of the Fourier components separately.

All of these considerations can be compactly stated using the terminology we have at hand. Taking the moving average is a convolution with a set of weights. The complex gain of the moving average is the Fourier transform of the coefficients. Filtering corresponds to multiplying by the complex gain in the frequency domain.

This has given but a small introduction to the analysis of linear filters. It does not tell us how to design a filter to perform a desired task. For this problem, the reader is referred to general texts such as Hamming,³ or Blackman and Tukey,¹ or to more specialized works such as Kaiser.⁵

2. SECTIONING. After we have chosen the weights with which we would like to filter, we still must choose how to implement the filter. A typical situation might be to have 15,000 data points which we wish to filter with 50 weights. We could do this by computing the convolution directly. We might also extend our data to 16K by adding zeros and our filter to 16K weights by adding zeros and then doing two Fourier transforms to get our convolution. For this choice of numbers the decision is not dramatically in favor of one or the other method.

Applying a 16K Fourier transform to only 50 weights seems rather extravagant. Perhaps we could do better if we were to filter only sections of the original data at any one time as first suggested by Stockham.⁷ The problem is then to find a sensible size for the various sections. Let us say that we have D data points and F filter weights. We want to find N (the section length) so as to minimize the total time. This scheme will take about $D/(N-F)$ Fourier transforms (transforming two real sequences at one time).

The total time will then be $t = \frac{D}{N-F} cN \ln(N)$

where c may have a small dependency on N which we will ignore. By assuming that $F \ll D$ and that N is continuous we may find the minimum of t by differentiating with respect to N . Thus

$$\begin{aligned} \frac{\partial t}{\partial N} &= Dc \frac{\partial}{\partial N} \left(\frac{N \ell n(N)}{N-F} \right) \\ &= Dc \left\{ \frac{\ell n(N)}{N-F} + \frac{N}{N-F} \frac{1}{N} - \frac{N \ell n(N)}{(N-F)^2} \right\} \\ &= \frac{Dc}{(N-F)^2} \{ (N-F) \ell n(N) + N - F - N \ell n(N) \} \\ &= \frac{Dc}{(N-F)^2} \{ -F \ell n(N) + N - F \} \\ \frac{\partial t}{\partial N} = 0 &\text{ implies } N - F(1 + \ell n(N)) = 0 \end{aligned}$$

which yields

$$F = \frac{N}{1 + \ell n(N)}$$

For our case of 50 filter weights, this suggests using Fourier transforms of length about 300, for a reduction of the time required by a factor of two.

Interpolation. One of the standard operations in numerical computing is the interpolation in a table of function values to obtain approximate values on a grid at a finer spacing than the original.

1. **BAND-LIMITED INTERPOLATION.** The most obvious application of these ideas is to the case of band-limited interpolation, sometimes called trigonometric or cosine interpolation. This interpolation is done over the whole interval, by fitting a trigonometric polynomial (or complex exponential series) of sufficiently high order through all the given values, then evaluating it at the new points.³ But the first step of this is exactly the finding of the Fourier transform of the original data. And the second step, as we shall see, can be accomplished by inverse transforming the sequence whose low order coefficients are the same as the coefficients of the transform of the original data, and whose high order coefficients are zero.

To justify this last statement, consider the case where the function is given at N equispaced grid points. We wish to interpolate in this sequence to obtain values at M times as many points. The original sequence of values, $X(t)$, has a Fourier transform $X(\hat{t})$ such that we have the representation

$$X(t) = \frac{1}{N} \sum_{\hat{t}=0}^{N-1} \hat{X}(\hat{t}) e\left(-\frac{t\hat{t}}{N}\right)$$

Consider the augmented Fourier transform $\hat{Z}(\hat{s})$ $\hat{s} = 0, 1, \dots, Nm-1$ such that

$$(a) \quad \hat{Z}(\hat{t}) = \hat{X}(\hat{t}) \left(-\frac{N}{2}\right) < \hat{t} < \frac{N}{2}$$

$$(b) \quad \hat{Z}(-N/2) = \hat{Z}(N/2) = 1/2 \hat{X}(N/2) \text{ if } N \text{ even}$$

$$(c) \quad \hat{Z}(\hat{s}) = 0 \text{ for all other values of } \hat{s}.$$

When we recall that $\hat{X}(\hat{t})$ has period N and that $\hat{Z}(\hat{s})$ has period NM we identify

$$\hat{X}(-\hat{t}) = \hat{X}(N-\hat{t})$$

and

$$\hat{Z}(-\hat{s}) = \hat{Z}(NM-\hat{s})$$

The construction of \hat{Z} corresponds to taking the circle on which the Fourier transform coefficients of \hat{X} are defined, breaking it at $\frac{N}{2}$, and inserting $M(N-1)$ zeros so that the low frequency Fourier coefficients of the two sequences match, and the high frequency coefficients of $\hat{Z}(\hat{s})$ are zero. What is the inverse transform of $\hat{Z}(\hat{s})$?

$$Z(s) = \frac{1}{MN} \sum_{\hat{s}=0}^{MN-1} \hat{Z}(\hat{s}) e\left(-\frac{(s/M)\hat{s}}{N}\right)$$

which, we note, is just the band-limited interpolation of $X(t)$, except for a factor $\frac{1}{M}$. For example, when $s = Mt$,

$$\begin{aligned} Z(Mt) &= \frac{1}{MN} \sum_{\hat{s}=0}^{MN-1} \hat{Z}(\hat{s}) e\left(-\frac{t\hat{s}}{N}\right) \\ &= \frac{1}{M} X(t) \text{ by the definition of } \hat{Z}(\hat{s}) \end{aligned}$$

Applying this in the case where the original values are not at equispaced points merely makes the finding of the first transform, $\hat{X}(\hat{t})$, more complicated. Furthermore, the extension of this scheme to multiple dimensions is immediate.

We remark that a fundamental fact of band-limited interpolation is that it is periodic, so that the first points of a sequence influence the values interpolated between the last points. This property, and the long range effects of single values, may mean we should avoid band-limited interpolations for certain uses. It may, however, be extremely useful in other circumstances.

2. **GENERAL INTERPOLATION RULES AS CONVOLUTIONS.** Knowing the limitations of band-limited interpolation, we often prefer to use a local formula such as Lagrangian polynomial interpolation. Using any interpolation rule such that the interpolated

values are linear combinations of the adjacent known values to subtabulate a table of equispaced data may very conveniently be done by convolutions. If we denote the value to be interpolated at $t + p$, $0 \leq p < 1$, by $Z_p(t)$, then the series $Z_p(t)$ is the convolution of the original series $X(t)$ with an appropriate series of weights:

$$Z_p(t) = \sum_{s=\alpha}^{\beta} W_p(s)X(t-s)$$

It is possible to arrange that the convolutions for all values of p be done simultaneously, but the computational effort may be increased over doing them separately. Since, typically, the series of weights will be short compared to the length of the series $X(t)$, it may be profitable to employ the sectioning described earlier. When applied to problems in more than one dimension, additional profit may be made if we use interpolation rules which are direct products or direct sums of lower dimensional rules, for example, a two dimensional interpolation rule such that the weights $W_{pq}(r,s)$ are either a product $W'_p(r)W''_q(s)$ or a sum $W'_p(r) + W''_q(s)$. The advantage of such rules is that the higher dimensional convolutions may then be done as a succession of lower dimensional convolutions or equivalently that the Fourier transforms of such sequences are the products or sums, respectively, of the Fourier transforms of the constituent parts.

Cauchy Products for Symbolic Polynomial Manipulation. The Cauchy product of two infinite series is a well-known result of college algebra:

$$\sum_{i=0}^{\infty} a_i x^i \cdot \sum_{j=0}^{\infty} b_j x^j = \sum_{k=0}^{\infty} c_k x^k \text{ where}$$

$$c_k = \sum_{n=0}^k a_n b_{k-n}$$

subject to some convergence criteria. We readily recognize that the $\{c_k\}$ are convolutions of the $\{a_i\}$ and $\{b_j\}$ and that we could use our techniques to evaluate these discrete convolutions.

We may arrive at this by considering an alternative viewpoint. Let us write $z = e(\theta)$ and recognize that $z^n = e(n\theta)$. With this notation the Fourier transform becomes

$$\hat{X}(\hat{t}) = \sum_{t=0}^{N-1} X(t)e(i\hat{t}/N) = \sum_{t=0}^{N-1} X(t)z^t \text{ where}$$

$$z = e(\hat{t}/N)$$

This is a truncated power series evaluated at points on the unit circle in the complex plane. Our convolution theorem is now a statement about multiplying polynomials and using a Cauchy product representation. The inverse Fourier transform gives us a way of evaluating coefficients of polynomials given as values equispaced on the unit circle.

Of the applications so far suggested, this most deserves the epithet "fun." It is doubtful if this technique will supplant conventional symbol manipulation techniques as it makes no use of special properties of the coefficient sequences. It does, however, admirably illustrate that maintaining an open mind may well bring forth new, surprising and occasionally beneficial applications of finite discrete Fourier analysis.

Considering Problems in Transform Space

It has long been a fact that some problems of physics are more tractable in transform space than they are in the regular coordinates. Our newfound ability to switch from one to the other may mean that some previously intractable problems will now be solvable. In electromagnetic problems, the return signal can often be represented as a convolution with a kernel. Taking the Fourier transform can change the form of the non-linearity from convolution to multiplication and may make the problems more manageable. In pattern recognition problems, one may seek measures which are free from the effects of linear transformations. Taking the Fourier transform may greatly reduce the problem of finding such measures. It has been suggested that quantizing the Fourier transform of a signal rather than quantizing the signal itself may result in a higher quality transmission for communication.

There are certainly a wealth of problems in which Fourier analysis arises as a most natural tool to be employed. Unfortunately it has often been rejected because of its high computational cost. Perhaps the application of the fast Fourier transform could swing the economic balance to make this time-honored technique again competitive.

ACKNOWLEDGMENT

The authors would like to express their appreciation to those from whose work we have borrowed, and whose fruitful conversations have helped in the preparation of this paper. In particular we should

like to thank C. Bingham, A. B. Langdon, C. L. Mallows, T. G. Stockham, and J. W. Tukey.

REFERENCES

1. R. B. Blackman and J. W. Tukey, *The Measurement of Power Spectra*, Dover, New York, 1958.
2. J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, (1965) pp. 297-301.
3. R. W. Hamming, *Numerical Analysis for Scientists and Engineers*, McGraw-Hill, New York, 1962.
4. R. W. Hockney, "A Fast Direct Solution of Poisson's Equation Using Fourier Analysis," *Journal of the Association of Computing Machinery*, vol. 12, no. 1, (1965) pp. 95-113.
5. J. F. Kaiser, "Some Practical Considerations in the Realization of Linear Digital Filters," Proceedings of the Third Allerton Conference on Circuit and System Theory, Monticello, Illinois, October 1965.
6. G. Sande, "On an Alternative Method of Calculating Covariance Functions," unpublished, Princeton University, 1965.
7. T. G. Stockham, "High Speed Convolution and Correlation," *AFIPS, volume 28, 1966 Spring Joint Computer Conference*, Spartan Books, Washington, 1966.
8. Whittaker and Robinson, *Calculus of Observations*, Blackie & Son, London, 1944.
9. J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

A SYSTEM FOR AUTOMATIC VALUE EXCHANGE (SAVE)

Vern E. Hakola

Touche, Ross, Bailey and Smart, Los Angeles, California

and

Sherman C. Blumenthal

Union Carbide Corporation, New York, New York

INTRODUCTION

The central theme for a System for Automatic Value Exchange (SAVE), from which all other key features of the concept stem, is that the exchange of money and credit can take place independently of documentary instruments such as checks and vouchers. SAVE contemplates substituting an electronic financial clearing process for the major portion of these traditional means of exchange in the consumer economy. The purpose of the system is to provide rapid and paperless financial settlements among the parties in buying-selling and debtor-creditor relationships, thereby achieving the economies and the convenience that centralization of major portions of the credit functions for entire metropolitan areas (and eventually the national economy) would permit. SAVE would integrate the functions of purchase authorization, credit transfer, bill payment, short-term lending, payroll, budget management and related elements.

The routine flow of payments for goods and services within such a system of exchange would become relatively instantaneous. Thus, verification of the payer's authorization and ability to pay must occur

as first rather than last step in the normal financial clearing process. Payment would, in fact, take place simultaneously with the receipt of goods or services.

It is hypothesized that SAVE would achieve a net economic gain and a dramatic improvement of service:

1. For associated banking institutions through increased deposit and loan relationships, elimination of float, decrease in amount of cash and negotiables outside the bank, and the creation of profits from new services.
2. For organizations purveying goods and services to consuming individuals and organizations through substantial reduction of financial data processing burden, elimination of bad debt losses, maximization of liquidity, and increase of credit purchasing activity.
3. For consumers through convenience of "universal" credit purchasing, budget management, reduction in total cost of credit, elimination of bill paying and check writing, and reduction in costs

of goods and services through lessening of variable overhead expenses in connection with their sale.

SAVE assumes that the role of the banking system will greatly expand in the future as compared to its role today. A SAVE bank complex would act as a buffer between debtor and creditor. Most accounts payable and receivable would no longer exist as we know them today. Any excess of receivables over payables would be reflected instantaneously as additional deposits to the SAVE bank complex; excess of payables, on the other hand, would represent a loan. Traditional charge accounts would be replaced by demand accounts supplemented by automatic overdrafts. Loan relationships with retailers of goods and services would include all forms of commercial lending.

Thus, the first area of expansion for banks would be in the area of short-term credit, both personal and commercial. At the present time, only a small part of the short-term credit needs of the economy are met by banks and other financial institutions. The larger part consists of "internal" credit in the form of accounts receivable and prepaid expenses, offset by accounts payable and accrued liabilities. SAVE visualizes immediate settlement of most routine obligations. Theoretically then, a fully realized SAVE would neither expand nor contract overall credit requirements. As it relates to fiduciary entities, SAVE would cause either a gain or loss of cash as compared to the situation otherwise.

In the field of personal credit, some banks have already established revolving credit for individuals. This takes such forms as "Ready Credit" at First National City Bank in New York, and overdraft banking exemplified by the "Ready Reserve" system at Security First National Bank in Los Angeles. SAVE would involve an expansion in this area, particularly in the latter direction. That is, personal credit now taking the form of various kinds of charge accounts at individual retailers would be replaced by bank overdraft facilities.

SAVE would minimize the preparation, processing and movement of documents within the banking system and the money economy at large, particularly the retail portions of it. Float, of course, would be largely if not entirely eliminated. Payer and payee would know at the time of a sales transaction that payment had, in fact, been consummated. The functions of instantaneous purchase authorization, automatic bill payment, automatic credit injection, con-

sumer and retailer budget management, "instant cash," checkless-cashless payroll, etc., would have strong economic attractiveness for nearly all subscriber groups, as well as providing means of solving many of the potentially competitive situations posed by existing credit bureaus, credit plan companies, existing bank services and retailer credit plans.

The ensuing sections of this paper are devoted to discussion of feasibility and a technical and functional description of SAVE operation. The service will necessarily involve utilization of a very large capacity communication network, a large number and wide variety of remote input/output devices graduated in complexity from simple push-button telephones to satellite computers, a foolproof means of personal identification, the use of a common "financial" number system, large and fast random access storages, and computers.

What are some influential factors that have emerged which account for the trend toward the emergence of a system such as SAVE?

While the economy has been expanding at a tremendous rate, the use of banking services such as demand deposit accounts is expanding even faster, as seen in Fig. 1. Indeed at present rates of accelerating growth, check clearings will more than double in 10 years.

Secondly, since World War II, banks have been under continuing pressure from competing institutions and their own customers to provide a variety of new customer services, both for businesses and

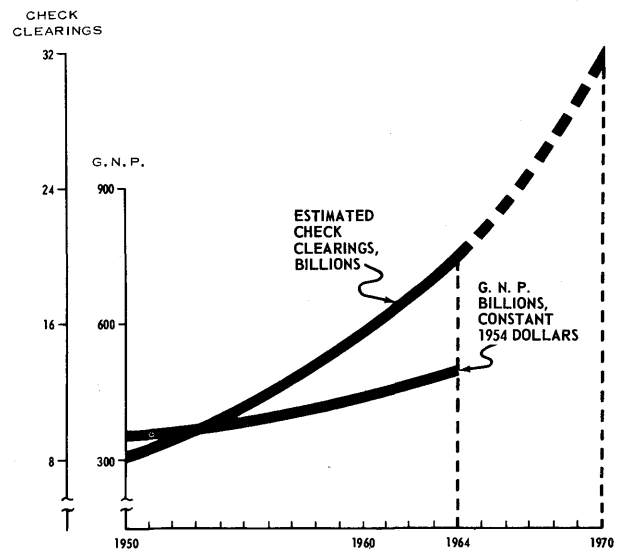


Figure 1. Comparison of check earnings and GNP forecast, 1950-70.

individuals. An automatic financial clearing process such as SAVE can be viewed in part as an extension of this trend. Moreover, it has so many ramifications in other parts of the bank's business that it must be viewed not merely as a new service and a new source of profit, but as dramatically supporting of the traditional loan and deposit relationships of the bank.

A third factor, of course, is the technological feasibility of SAVE as a result of major industry advances. The technology involved in SAVE, which combines computers, communications, and special terminal devices, is implemented in and familiarly illustrated by the American Airlines reservation system known as SABRE.

Fourth, there is the emergence within banks and the organizations with whom banks deal of professional systems people of a new breed, who are not limited in their thinking by the traditional boundaries of their trade. As business relationships change, as new techniques come to the fore, these professionals are pressuring their own managements to exploit the opportunities arising from the technological fallout taking place now almost daily.

A fifth factor is the explosive growth in personal and installment credit. The growth, universal acceptance, and use of credit by individuals is well known to bankers and retailers. Today, there is a growing need for control in the extension and use of personal credit. Banks have a major opportunity in the short-term consumer credit area which they are not now fulfilling. It is often too frequently true that an individual's multiple credit lines are controlled by nothing other than his own good sense. Banks have a responsibility in this since they ultimately underwrite the extension of much of this credit, directly and indirectly. The costs, the number of accounts, the cards, the statements mailed, credit checks, etc., are multiplying in a seemingly profligate and unnecessarily burdensome way.

In summary, we believe that SAVE will emerge to meet the eventual necessity for improvement in the value exchange environment which is illustrated in the above factors. Therefore, it is clear that banks of all sizes must do everything in their power to begin to establish themselves more firmly at the center of the consumer-merchant-vendor communication channels, and thereby strengthen their relationships among these major economic sectors.

These factors have apparently been influential in helping a number of thoughtful people in the com-

puter professions to reach similar conclusions.¹⁻³ Influential people in the banking industry, too, are no less convinced that the SAVE concept should take its place in the forefront of their future planning.⁴⁻⁹

SYSTEM FUNCTIONS

Balance information in the consumer's demand deposit account at his bank must be accessible directly to the point of sale in order that the initial system function—purchase authorization (or verification of the ability to pay)—can be accomplished. If there is sufficient balance in the account to cover the amount of the contemplated transaction, authorization would be automatic. If for any reason the purchase cannot be automatically authorized, a human decision-maker would enter the loop in the form of an authorizer who would examine the consumer's credit history, and discuss the problem with him on the phone before deciding finally to authorize or refuse.

In the event there is insufficient balance in the consumer's account to cover the purchase, credit may be automatically injected into his account by his bank on the basis of a prior revolving loan arrangement, provided he was not delinquent or over-limit. This would be SAVE's counterpart of the traditional retail revolving or option account. The source of these overdrafts may be a debit made at the time of sale, or could arise as a result of other kinds of transactions the consumer has authorized SAVE to make payment on; as utility bills, rent bills, installment loan payments, automatic savings deposits, etc.

The transfer of money between the accounts of payers and payees, or between separate accounts of the customer, or between the bank's lending pools and its depositors can take place within a single bank or between separate banks who are participants in SAVE. There would, therefore, no longer be any handling of foreign items, interbank clearings as such, no float, NSF's, holds or stops. The transfer would not generate an accounts receivable, thus eliminating a major cost to the merchant. A large portion of what are credit purchases today would be turned into essentially cash transactions.

It is not, of course, practical to eliminate cash entirely. When the amount involved is quite small, buying a newspaper, for example, or where non-SAVE subscribers are involved, it would be difficult

to avoid using cash. In order for a consumer to obtain cash conveniently, especially in view of the probability that he would no longer make regular visits to the bank on payday, SAVE would here provide for a means of authorizing the transfer of cash at prescribed locations. This might be looked upon as a public service to be performed for the consuming public as a means of keeping a small but essential part of the economy operating. Such "instant cash" locations (if we may call them that) might be operated at employer's premises, at bank branches, in transportation terminals, or even at special service points established for this purpose by SAVE.

Automatic bill payment, unlike an unanticipated transaction requiring authorization at the point and time of sale, would require some form of pre-authorization on the part of the payer. Sometimes such pre-authorized payments would be for a fixed amount, like a monthly rent bill; in others, where the amount would vary as in the case of a utility or phone bill, payment would be made automatically on presentation up to a fixed amount. The invoice need not be a visible document, but could equally as well be an electronic communication generated at the payee's premises by his computer system, if there is one, or by a billing operator seated at the keyboard of an on-line billing machine interconnected with SAVE. If SAVE, for any reason, was unable to make payment upon presentation, notice of nonpayment would be immediately transmitted to both parties for appropriate action.

Payroll is a primary source for the generation of consumer funds, and becomes the basis for essentially all consumer activity. Consequently, the payroll process exerts a key influence over the adequate functioning of SAVE. If the net pay of a consumer is automatically credited to his account through the operation of SAVE, the credit reliability of this individual will become more firmly established and permit the network to forecast the flow of funds into his account, thus assuring the constancy of a consumer's willingness or ability to generate funds to cover his economic activities. This increased capability of forecasting income against expenses would enable SAVE to project that flow and to avoid over-limit situations. The system might offer payroll computation service to those employers who so desire. In any event, the employer's account would be debited the total net pay, and each employee's account credited with his net pay.

Often, the overdraft revolving type loan arrange-

ments to cover short term credit needs of the consumer will not be suitable for large purchases such as automobiles or major appliances. These special types of consumer loans would be negotiated by the consumer with the bank as at present, and proceeds would be credited to his account. SAVE could see to it that the account was periodically debited for the installment amount and could utilize the overdraft facility for the purposes of making these installment payments, if necessary.

Other functions that could be performed by SAVE for its subscribers, once the basic kinds of services just described were in operation, could be based on the fact that the system would have available certain comprehensive and up-to-date financial data on subscribing merchants, utilities, service organizations, professionals, and others with whom a consumer would deal, and would also have financial data on the consumers themselves. It would, in fact, have the basis for moving into revenue accounting, into cash and budget management, and even into tax services. More speculative perhaps, is the potential role of SAVE as a source of statistical and consumer marketing information.

Figure 2 illustrates how the several functions just described are integrated in SAVE. Note that the SAVE computer does not supplant the computers installed in the several bank participants. It only supplants the clearing house function as well as the transit functions now handled in the bank. It acts, moreover, as an interchange among the banks, retailers, employers, public utilities and so on.

The central theme on which this integration is based is the potential for instantaneous transfer of credits for the majority of transactions in the community. In order to accomplish this, we have seen that it is possible and desirable to eliminate the check, eliminate the deposit and eliminate the documentary processes that underlie the generation of these pieces of paper. We have seen that automatic value exchange involving a multitude of small transactions requires that verification of the ability to pay be routinized at the point of sale. Without this facility, the banks participating in SAVE expose themselves to unacceptable risks.

Of course, this entails other things. For example, it is difficult to justify the equity in authorizing a transaction without assuming the responsibility for collecting the obligation incurred by the consumer as a consequence of the authorization. The authorizer is, in effect and by implication, the guarantor.

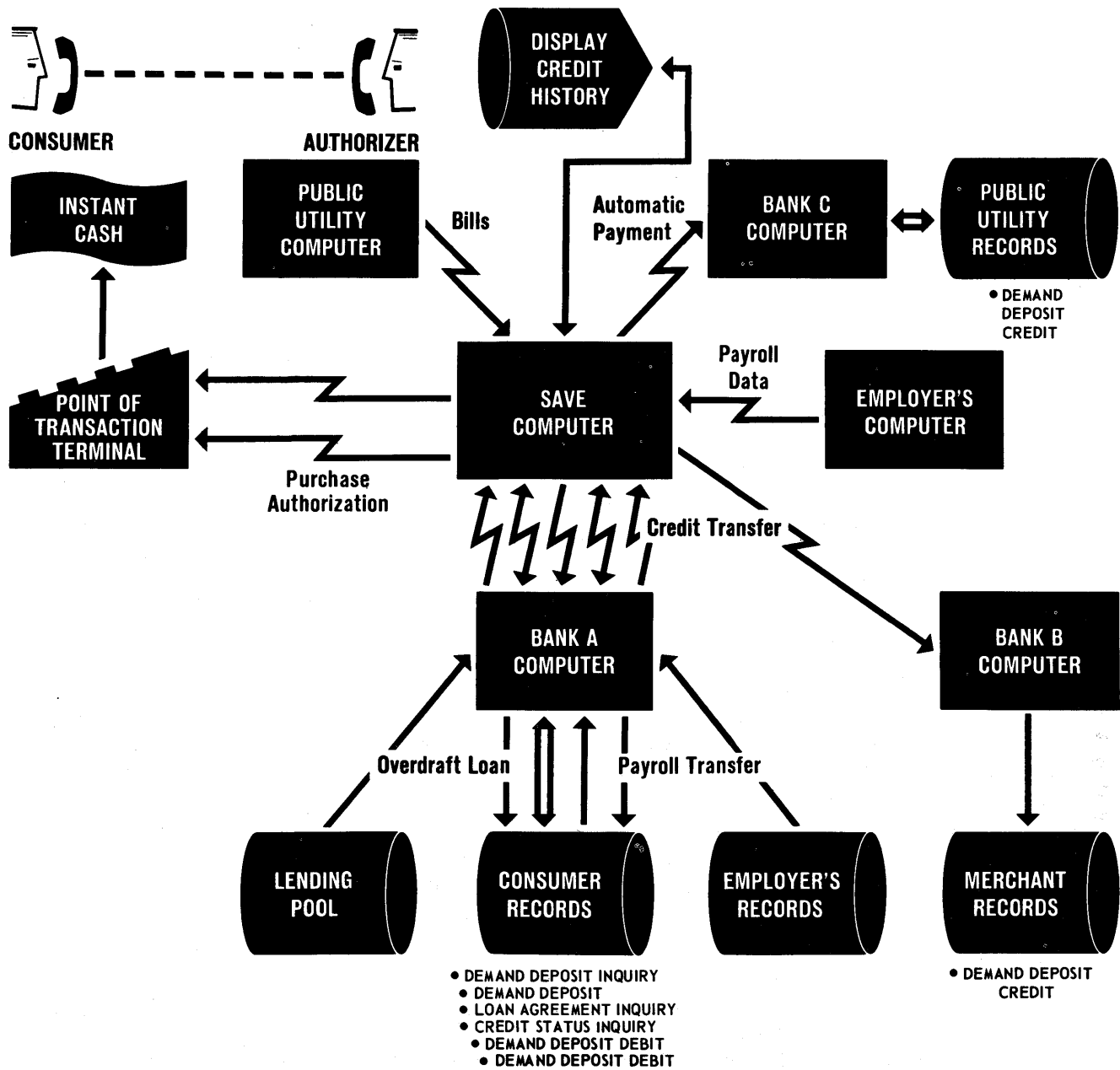


Figure 2. Diagram of the SAVE system.

As such, the authorizer (in this case SAVE) is performing the equivalent of financing the retailer, by providing him with immediate credits to his account. Therefore, we see that these two functions—credit transfer and purchase authorization—go hand in hand and cannot be considered independently. Revolving overdraft loans also fit the requirements of an essential function. Without this overdraft banking feature, the whole operation of automatic value exchange may very well grind to a halt amid a welter

of irritations at its inadequacy in meeting the day-to-day and hour-to-hour needs of the consumer. Therefore, “overdraft banking” as we have described it is a practical necessity for the operation of the whole system. It is a type of lending activity that will make routine consumer activity possible at all times, even though it may not cover unusual purchases, such as automobiles and homes.

Next, for several reasons automatic bill payment appears to be an essential service ingredient. First,

because of the sheer volume of payments involved. Second, these payments consume a major share of the consumer's income. Third, the payments are almost always today paid by check or money order rather than cash. But the overriding reason is that if checks were continued in use for the purpose of making bill payments and were tendered for major amounts of the consumer's account balance, and if this is the same account balance with which SAVE operates, then the consumer would somehow have to be aware at the time of tendering the check that his balance is adequate. Thus, the consumer would have to be able to make an inquiry of the system to obtain current balances and then to subtract outstanding checks not yet presented for payment to obtain available balance. This is obviously cumbersome and self-defeating. The alternative of a checking account separate from his account with SAVE is even more self-defeating. Thus, we are suggesting very strongly that the inclusion of automatic bill payment is an essential feature to the satisfactory operation of SAVE.

Looking again at Fig. 2, this time at the input side, it can be seen that the major source of funds supporting all consuming activity is the automatic, periodic payroll deposit. We have argued that SAVE would operate more soundly if it were known as a matter of course that the net pay of its consumers was automatically available at given intervals. If, to the contrary, a consumer might optionally not deposit his pay, or deposit arbitrary and varying amounts of his pay at arbitrary and varying periods, SAVE would obviously have to set a lower limit on overdraft credit to the consumer. The consumer, therefore, could retain final authority over his salary disbursement only at the expense of reduced flexibility in his other financial and consuming activities.

Given the degree to which it is theoretically possible to impose automaticity in the exchange of value in the consumer economy, there remains the vital matter of routinely and comprehensively keeping the individual consumer informed of the personal economic consequences of his consuming behavior. Today, information in the form of bank and charge account statements, telephone bills and the like, are to some extent confirmatory in nature, and serve to enable the consumer to reconcile his own records. In the era of SAVE operation, a much more powerful, concise and comprehensive message must be generated to keep the consumer informed of the results of

what he has done and its economic implications for him.

Figure 3 suggests how this information might be presented to the consumer routinely. It combines features found in the checking account statement, the descriptive (rather than "country club") department store bill, single unit purchase invoices and loan account take-down and repayment information. It is weakest in handling multiunit purchases such as phone calls—the details of which would probably still have to accompany the SAVE statement as an attachment. Special notice should be taken on the SAVE payment on 4-16, the SAVE take-down on 4-26 (lest the account go negative), and the summary of overdraft account status in the lower of the two top lines.

This, then, is the interplay of the various functions that we have described in the operation of SAVE. They show the intimate interdependence of these functions, and they together provide SAVE with the powerful impact of their combined operation.

DESIGN CONSIDERATIONS

Even though community-based SAVE systems may go into operation over a period of years in many different localities, these systems should be designed in a manner which would permit their eventual interconnection. Thus, account numbering schemes, terminal performance specifications, file record contents, credit scoring procedures, merchant

FIRST NATIONAL BANK					ACCT. NO.	
St. Francisburg, U.S.A.					312-20-8479	
Edward G. Smith					STATEMENT DATE	
1421 Adam Ave.					5-13-69	
St. Francisburg, U. S. A.						
PREVIOUS DDA BALANCE	281.05		DEBITS	CREDITS		CURRENT DDA BALANCE
	NO.	AMOUNT	NO.	AMOUNT	HANDLING CHARGE	
	17	590.08	3	551.90	3.60	241.40
PREVIOUS SAVE BALANCE	180.00		OVERDRAFTS	PAYMENTS		CURRENT SAVE BALANCE
			50.00	60.00		172.70
					INTEREST CHARGES	
					2.70	
DATE	DESCRIPTION		DEBITS	CREDITS	BALANCE	
4-11					281.05	
4-11	Univ. Ret. - Appliance Dep't		100.00		181.05	
4-12	ABC Co. - Salary			250.95	432.00	
4-16	SAVE - Payment		62.70		369.30	
4-17	Elec. Co. - Utility Bill		24.25		345.05	
4-17	Bell Co. - Phone Bill		15.30		329.75	
4-18	Acme Elec. - Repair		25.00		304.75	
4-20	Ace Mort. - Mortgage		151.75		153.00	
4-21	A&P - Food Purchase		37.50		115.50	
4-22	M. S. Smith - Physician		10.00		105.50	
4-26	SAVE - Overdraft			50.00	48.25	
4-28	ABC Co. - Salary			250.95	299.20	
4-30	A&P - Food Purchase		14.20		285.00	
5-3	SAVE - Cash		20.00		265.00	
5-5	Met. Op. - Entertain.		10.00		245.00	
5-7	FNB - Service Chg.		3.60		241.40	

Figure 3. A possible consumer statement from SAVE.

billing fees and schedules, line item descriptors, input/output data formats, and numerous other elements of the system design should be standardized to the fullest extent possible. Such standardization will ultimately be of benefit to:

- Consumers, who will be assured of uniform treatment wherever SAVE operates. (An important point of strength in present-day Diners' Club, American Express and similar national credit card plans.)
- Merchants, utilities and other business subscribers, who will be assured of a reasonably uniform fee structure and level of service regardless of the locus of their customers or their branches.
- SAVE itself, from the standpoint of uniformity and economy of systems design and programming, and ease of operation.

These problems of standardization are admittedly enormous, but cannot be regarded as insuperable. Standardization pockets have already begun to appear to satisfy needs other than those of SAVE, and this standardization will undoubtedly broaden in future years (e.g., one can expect the use of social security numbers as an identifier to continue to increase).

The magnitude of the standardization task depends importantly upon the nature of the controlling forces behind SAVE. If one agency could specify uniform equipment and terminal procedures and also specify the merchant, banking, and other system interfaces, this would represent the most straightforward and controllable system design environment. If, on the other hand, the best that could be achieved would be a loosely federated network of local systems, with the equipment and system control specified locally, and without a strong coordinating authority, this would constitute the worst-case condition.

The remainder of this section is based upon tacit assumption that the systems environment can be controlled to the extent that a workable degree of uniformity on a national scale is achievable.

Universal Numbers

With respect to the consumer, there is little doubt that the social security number (taxpayer identification number) represents the wave of the future. The

banks, key components in the SAVE system, already possess this number for their savings account customers for the purpose of reporting interest income. Many banks are now seeking, in their new generation direct access systems, to consolidate all the account relationships of a single customer under one number, and the social security number is the one most readily at hand. Thus, upon input of this master number, an individual's demand deposit, savings and loan account could all be retrieved from the bank's direct access data base in real time.

This is not to suggest that the banks will immediately abandon their present account numbering systems or that they can no longer accomplish a name retrieval. These other customer identifiers may, however, assume secondary significance, with possibly a cross-reference to the master social security number.

Recently, Iowa became the first state to convert its driver license system to social security numbers. Now, several other states have decided to, or are considering following suit. This, coupled with the already heavy federal government and banks usage, seems to reinforce a developing trend.

The taxpayer identification number is also available and applicable to merchant establishments and other businesses and may for the purposes of SAVE prove to be most efficacious here as well as with consumers. The "DUN's" numbering system (developed by Dun & Bradstreet) lacks universality in that it applies to business organizations only, and does not extend to municipalities, school districts, charitable organizations, churches, and the variety of other institutions which might participate in SAVE.

Consumer Identification

Historically, there have been two divergent systems approaches to the problem of consumer identification:

1. Issuance of an identification card containing descriptive and other identifying data which offers prima facie evidence that the individual seeking to use the card is, in fact, the card owner. Typically, such cards contain signature, physical description, and sometimes a photo or fingerprint. The automobile driver's license is perhaps the best example of such an approach to identification.

2. Issuance of a card which contains no descriptive information: the identification being accomplished by means of a code or number on the face of the card. The assumption underlying this type of card is that the bearer, merely by virtue of his possession, is entitled to use it. Examples of this approach can be found in certain gasoline credit cards and industrial security badges.

The advent of SAVE will introduce a whole new dimension into the question of identification in general and the use of identification cards in particular. Heretofore, the card was constituted evidence that its possessor has passed a credit check and, as a consequence, has been granted a pre-authorization to obtain goods and services on a credit basis. The credit check must necessarily *precede* the card transactions in time and must be structured to *anticipate* the reliability of the card user's purchasing and repayment proclivities.

Such a situation obviously implies a rather loose system of controls with an attendant exposure to loss. Losses can be reduced by more stringent screening—i.e., raising the approval threshold, but such action reduces the volume of transactions and increases the cost of administering the system. Present day credit card plans endeavor to strike a balance between risk and profitability in various ways. BankAmericard, for example, sets an overall dollar limit for each card holder based upon an analysis of his disposal income.

Because of the pre-authorized nature of today's cards, they continue to be "live" enough even though the owner's credit status may have changed since the card was issued. For this reason, most credit cards such as Diners' Club, American Express, and BankAmericard are reissued frequently. In some of these plans, the expense is incurred each quarter. Lists of bad or stolen cards must be distributed to the participating merchants, who must endeavor to screen these out in the course of each credit transaction.

With SAVE, this entire picture is changed. The card, if indeed a card is to be used, is no longer so much a "credit" card as a true identification card, since a transaction processed through the system may or may not involve credit. Depending upon the nature of the transaction and the user's preference, the card may simply authorize the system to effect a transfer of funds and credit transactions when they

do occur, and are no longer based upon an earlier credit evaluation since credit status can be reviewed in real time at the occurrence of each transaction. System controls are such that there is no particular reason why every member of a community should not be issued a card, *regardless of their credit status*, thus enabling them to participate in those facets of the system for which they may be eligible. In essence, the financial status of the individual under SAVE resides in the computer files, *not* in the card. Therefore, cards need not be reissued and the policing of lost and stolen cards becomes, upon notification of the system, immediate and absolute.

Among the various machine-sensible cards now in existence are three major types, each of which can prove suitable for SAVE:

- Hollerith punch coded card, such as the 18-column cards used with the IBM 1001.
- The Addressograph-Multigraph bar-coded card, presently used in off-line gasoline card applications.
- A modified version of the telephone company dialer card. These cards use a two-out-of-eight code punched with round holes and have square sprocket holes on two of their edges. Cost of cards such as these would vary depending upon material quality, presence of embossing, color treatment, etc.

Heretofore, there have been two standard sized cards:

- The "Mister Card" size issued by Diners', Carte Blanche, Unicard, etc., and all the oil companies.
- The "Mrs. Card", a longer, narrower card issued by retailers.

There is presently a serious standards problem engendered by the telephone company, whose roundholed cards offer tremendous potential for use in conjunction with Touch-Tone dialing equipment. Unfortunately, the telephone company's cards are slightly larger than a standard credit card. This means that either the Bell System must redesign its card or manufacturers of card producing and card sensing equipment, like Dashew, will have to modify their equipment.

Numerous other card identification schemes have been proposed, ranging from Martin Greenberger's

“money card”¹ to cards with machine sensible material, such as magnetic particles, imbedded in them. The most promising of these ideas need to be carefully investigated.

Beyond the use of a card for identification—and it is by no means certain that a card is the most appropriate means—numerous other approaches suggest themselves. Among these are:

- Signature recognition, in which the consumer's signature is captured and transmitted to the computer.
- Fingerprint scanning, in which the finger is placed on a sensitive pad built into the terminal device for on-line comparison by the computer.
- Voice recognition, in which a voice “signature” is transmitted to the computer.

There are numerous problems in the development of these methods, such that their practical employment is still several years away. Even farther away, but still conceivable, is true pattern recognition of the consumer's facial characteristics.

Terminals

According to some theorists,¹⁰ the time is approaching when the consumer might pay his bills and effect other forms of transfers of funds through use of a touch-tone telephone instrument located in his home. This prognosis seems doubtful for the foreseeable future, not so much because of the unavailability of touch-tone switching (a touch-tone “pad” can be attached to any rotary dial instrument) so much as because of the lack of a hard-copy transaction trail, and because of the problem of indubitable consumer identification at unsupervised locations.

Production of hard-copy output at the location and time of the transaction appears to be essential to SAVE from both an operational and a legal standpoint. This hard-copy capability can be very primitive, perhaps no more than a computer-actuated “stamp” that would certify for both parties that a computer-recorded transaction had taken place. Hard copy so generated would be retained by the merchant and consumer as a visible record to facilitate adjustments, to aid in bookkeeping between statement cycles, and to help the consumer reconcile his statement when received. This is similar to the documents produced manually under present-day credit card plans with the revolutionary exception

that paper generated by SAVE is external to and is not part of the mainstream of system processing.

One lawyer has gone so far as to suggest that a printed output at the computer center might have to be produced; at least in the initial stages of operation, to satisfy the courts that the action taken by the computer was, in fact, the same as that indicated by the response produced at the terminal. This is roughly akin to the journals produced by today's demand deposit accounting systems.

SAVE terminals must also have the capability of sensing and transmitting the customer identification number from an identification card, assuming that no more practicable means of identification is developed. Experience at Banker's Trust,⁶ with their tellers keying in account numbers for current balance inquiry purposes, indicated such a high degree of accuracy that it is conceivable that the customer identification card might not have to be machine sensible. This is a question to be kept in mind for subsequent study.

A method of terminal operator identification must be provided, through hardware, software, or a combination of the two. Possibly the identification card reader could be used to sense a special operator card used by the operator in a sign-in/sign-out procedure similar to that employed today in some airline reservation systems.

Confidentiality and Accountability

There is an understandable history of sensitivity on the part of banks about disclosure of account information, which is one of the reasons for the failure of some organizations' previous attempts to establish an automatic credit mechanism. Some of the bank's scruples have, in recent years, been overcome with the development of computer-sharing by the smaller banks for demand deposit accounting. Here, the accounts of a bank are maintained either by a large correspondent or by a cooperatively owned data processing center. The controls over the confidentiality of information by these centers should provide a good point of departure for the design of SAVE controls.

A potentially more serious aspect of the disclosure problem lies not in the area of one bank or business obtaining information about its competitors' accounts, so much as in the area of accessibility of information to unauthorized persons. Here, SAVE controls must be unbendingly strict and rigid,

so much so that, if for no other reason, the idea of a consumer using his touch-tone phone at home to communicate with the computer loses much of its appeal.

And, just as in other areas of potential abuse and defalcation within the traditional exchange system, the weakest links are those employees who interface most closely with the system; in this case the terminal operators and others who would have access to the SAVE files. A point of entry must be provided to accommodate those customers who do not present their identification cards. This means that a procedure to assure accountability to this and similar cases must be established.

There will need to be a set of terminal control records maintained in the SAVE files for accountability (and probably billing) purposes. The record for each terminal will identify the accountable operator and will contain operating standards and threshold limits which, if violated, will result in immediate notification of the operator's superior. Examples of tests on data in the terminal control records might include:

- Number of high dollar amount transactions in a given period.
- High, overall activity rate.
- Large number of cash-paid transactions.
- Unusual number of debit/credit reversals.
- Several successive transactions against the same account at the same terminal.

Besides controlling at the terminal level, controls must also be built into each consumer's record to identify runaway withdrawals of revolving credit and other abnormal account activity patterns. Attempts to maintain float between accounts (even accounts in different cities) should also be detectable.

CONCLUSION

This has been primarily a technical discussion. As such the vital issue of economic feasibility has not been touched upon. One highly knowledgeable proponent of automation in banking¹¹ has gone so far as to suggest that the economic rather than the technical issues should be the paramount concern of those who advocate systems such as SAVE. While this is a correct view, it fails to acknowledge that technical breakthroughs and new system conceptual-

izations can radically alter the point of economic return.

Assuming, if we may, the economic attractiveness of some more perfect configuration of SAVE elements, a major problem will be entrepreneurial. Rather than assuming that an automatic credit clearing network will be an inevitable, spontaneous development, it is going to take a whole new order of enterprise and organization and concentration of resources and deliberate planning. This is probably going to have to stem from some enterprise that is constituted just for the purpose of bringing this about.

Who might this be? There are numerous possibilities as we have seen. It could be an existing organization such as a national or local credit card plan; it could be a cooperative of banks formed for this purpose; it could be an independent entrepreneur moving into the community; it could be a nationally based entrepreneur who will franchise this in various communities and groups around the country; it could be a computer manufacturer; it could be an existing local clearing association whose scope of operations is expanded to encompass the realization of this kind of network; it could even be the Federal Reserve System.

Our guess is that the major initiative is likely to stem from the computer manufacturers. The reasoning is this: If our computer manufacturer market is going to change over the next several years from primarily the sale of hardware to individual users, to the sale of services in the form of very large information utilities of various kinds, then it will be vital to the competitive survival of the computer manufacturers that they take an active lead in some very important way in the establishment and operation of such utilities in order to maintain their income base. They are also in the best position to accomplish the technological development, to impose standards, etc., but even the largest of the computer manufacturers do not have the financial resources to go into the banking business on the cosmic scale that would be necessary to underwrite the cost of short-term credit of whole regions and whole communities, and of the whole country. That is why, even if our speculation is correct, the computer manufacturers are going to be prime movers in this development, they are going to need the close cooperation of financial institutions; and among the most logical candidates for participating with them in a cooperative endeavor are the commercial banks.

REFERENCES

1. Martin Greenberger, "Banking and the Information Utility," *Computers and Automation*, Apr. 1965.
2. Robert V. Head, "The Checkless Society," *Datamation*, Mar. 1966.
3. Arthur S. Kranzley, "The Bank of the Future," *Datamation*, July 1965.
4. Elizabeth M. Fowler, "Personal Finance: Juggling Checkbooks," *New York Times*, Sept. 28, 1964.
5. Merle E. Gilliland, "Banks Can Be Computer Utility Centers," *American Banker*, May 25, 1965.
6. Putnam W. Livingston, "The Stopping of Moving of Checks," *Computers and Automation*, Apr. 1965.
7. David C. Melnicoff, "Bank Management and the Marketing Concept," American Bankers' Association, Marketing Research Workshop, Mar. 18, 1965.
8. George W. Mitchell, "The Impact of Automation on Bank Structures and Function," *American Banker*, Dec. 30, 1965.
9. Anthony Oettinger, "The Coming Revolution in Banking," *Proceedings of ABA National Automation Conference*, New York, 1964.
10. L. Davidson, "A Pushbutton Telephone for Alphanumeric Input," *Datamation*, Apr. 1966.
11. A. R. Zipf, "A Practical View of Universal Credit," *Datamation*, Mar. 1966.

REAL-TIME RECOGNITION OF HANDPRINTED TEXT *

Gabriel F. Groner

*The RAND Corporation
Santa Monica, California*

INTRODUCTION

During the past 10 years, there has been considerable interest in the automatic recognition of hand-written and machine-printed symbols.¹⁻⁷ Most of these studies have involved computer recognition of an already completed symbol; a few have involved special electronics for the analysis and recognition of characters⁸ or words⁹ as they are being written.

Only recently, with the advent of on-line computer systems, have computers been used for the real-time recognition of handprinted symbols.¹⁰⁻¹³ Aside from Bernstein's recent work,¹³ real-time schemes to date have been concerned only with the recognition of single, isolated symbols. None of the real-time schemes (to the author's knowledge) have been used in a problem-solving environment, or have utilized contextual information for recognition.

This paper describes a symbol recognition scheme which allows an on-line computer user to print text naturally, and have it recognized accurately, as he prints it. The scheme responds very quickly even though it recognizes a fairly large set of symbols. Moreover, it imposes few constraints on style, speed, or position of writing. It makes use of contextual information to distinguish symbols which cannot be

distinguished by shape alone. This scheme has been used daily at The RAND Corporation for writing computer code, drawing flow charts, and editing. The symbol recognition scheme is written in IBM System/360 Assembly Language and runs on an IBM System/360 Model 40.

The symbol set consists of the upper-case Latin alphabet, the numbers, the symbols +, -, =, /, (,), [,], *, \$, ,, ,, ', ^, >, <, and a scrubbing action which causes erasure of underlying symbols. The scheme also recognizes a set of flow-charting symbols, but these will not be discussed. When any other symbol is drawn, it is either identified as one of the above, or as a "cannot interpret." This symbol set is sufficiently large to enable a user to communicate data and directives to a computer by using *only* a pen-like instrument.

The scheme is designed so that the user can concentrate on his problem, rather than on extraneous operational mechanics. It therefore accommodates a variety of printing styles, allows for the printing of several symbols in quick succession, and responds quickly. Finally, any errors made by the scheme may be corrected easily.

USER INTERACTION

A user communicates with the computer via a RAND tablet¹⁴ in conjunction with a cathode ray tube (CRT). The tablet hardware consists of a

* This research is supported and monitored by the Advanced Research Projects Agency under Contract No. SD-79. Any views or conclusions contained in this paper should not be interpreted as representing the official opinion or policy of ARPA or The RAND Corporation.

horizontal 10-in square writing surface and a pen-like writing instrument. Figure 1 depicts a user interacting with this hardware. As the user moves the pen near the tablet surface, a dot on the CRT follows the pen motion—this direct feedback helps him position the pen for pointing or drawing. When he presses the pen against the tablet writing surface, a switch in the pen closes, thereby notifying the central processing unit (CPU) of a “pen-down” action. As he moves the pen across the tablet, the pen’s track is displayed on the CRT—the pen “point” thus seems to have “ink.” When the pen is lifted, the pen switch is opened, thereby notifying the CPU of a “pen-up” action, and “inking” ceases. A new user easily adjusts to watching the pen motion on the CRT while actually moving the pen off-screen on the tablet. Furthermore, he finds it convenient to have no part of the CRT “working surface” covered by his hand or pen.

The user must print plainly; i.e., not use curls or scrolls, and not drag the pen between strokes. The letter O must be written with a slash to distinguish it from the number 0, the letter I with serifs to distinguish it from 1, and the letter Z with a crossbar to distinguish it from 2 (these conventions are commonly followed when writing on coding forms). The user may otherwise print in his normal style. When the user’s “inked” symbol is recognized, it is replaced by a hardware-generated version. He may change this symbol at any time, merely by writing over it.

Although the user may write anywhere on the tablet surface, he must follow some conventions of size and position just as he would when writing on a piece of paper. The display format subtly conditions him by displaying a recognized symbol in a standard size (full-size symbols are about $\frac{3}{16}$ -in high) at the standard location (one of a possible 3570) nearest to the center of his “inked” version of the symbol. Since the user writes next to the standard-size symbols uniformly spaced along lines, he soon learns to adjust his printing accordingly. This symbol size and separation are about the same as those commonly required for printing on coding forms.

ANALYSIS OF THE DATA

The purpose of data analysis is to extract those features most valuable for discrimination among the allowable characters; i.e., those features which have consistent values over variations of a particular symbol, yet which have different values for different symbols. Since a single scheme recognizes the print-

ing of many users, any of whom may print sloppily, consistency is assured only if a symbol can be described by a few gross features.

The on-line nature of this recognition scheme enables processing of the data point-by-point as the pen is moved across the writing surface. In order to minimize time and storage requirements, therefore, the scheme extracts features as the data arrive.

The Data

Pressing the pen against the writing surface activates the symbol recognition scheme by indicating the start of a stroke. As the pen is moved across the writing surface, the recognition scheme is notified of its position every 4 msec. Finally, when the pen is lifted off the surface, the recognition scheme is notified that the stroke is completed. Each pen position is accurate to about 0.005 in. The hardware thus provides a very detailed—about 100 data-points—time-sequential description of each stroke.

Smoothing and Thinning

The scheme smooths the data by averaging a newly arrived data-point with the previously smoothed data-point, thus reducing the noise due to

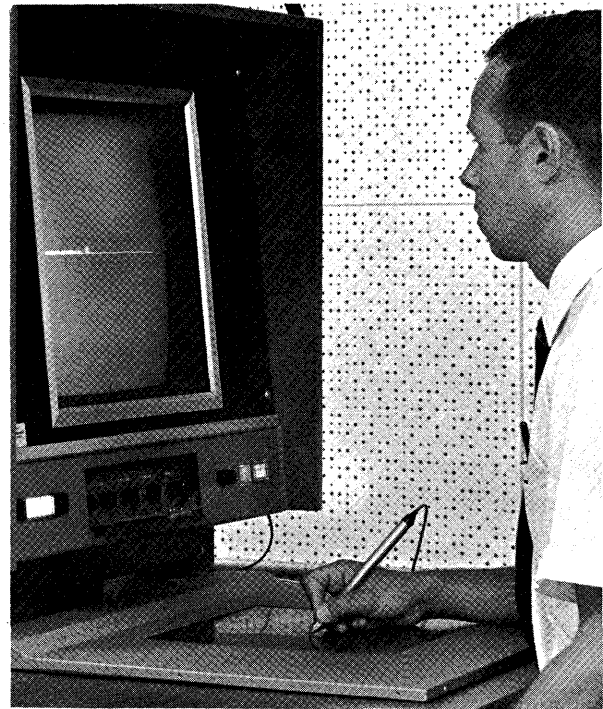


Figure 1. A user interacting with a RAND tablet and CRT.

the discreteness of the pen location as measured by the tablet. Smoothing is based on the equations

$$X_{Si} = \frac{3}{4}X_{Si-1} + \frac{1}{4}X_{Ri}$$

and

$$Y_{Si} = \frac{3}{4}Y_{Si-1} + \frac{1}{4}Y_{Ri}$$

where X_{Ri} and Y_{Ri} are coordinates of the i th raw data-point, and X_{Si} and Y_{Si} are coordinates of the i th smoothed data-point.

Thinning is the process of removing some of the data-points from the pen track. This is accomplished by comparing the position of a new smoothed data-point with the position of the last point in a thinned track. If these points are sufficiently far apart, the analysis scheme accepts the smoothed point as part of the thinned track; otherwise, it is discarded. Thinning eliminates small perturbations in the track, and reduces the data processing requirements by drastically reducing (by a factor of seven or so) the number of data-points. Thinning is described by

$$X_{Tj} = X_{Si}, Y_{Tj} = Y_{Si}$$

if

$$|X_{Si} - X_{Tj-1}| \geq \Delta$$

and/or

$$|Y_{Si} - Y_{Tj-1}| \geq \Delta$$

where X_{Tj} and Y_{Tj} are the coordinates of the j th thinned data-point, and Δ is a parameter of the analysis routine. The recognition scheme, which expects symbols to be drawn about $\frac{3}{16}$ -in high, uses $\Delta = 0.02$ in. This value of Δ is large enough to eliminate insignificant data-points, yet small enough to maintain the essential characteristics of the track.

Figure 2 is a photograph of a display generated by a program which did the following:

1. Replotted without any processing each sampled point of a figure drawn at the tablet.
2. Magnified the original figure eight times.
3. Smoothed the track.
4. Thinned the track with $\Delta = 0.01$ in.

The Curvature Feature

Curvature is the most obvious track characteristic which is independent of position and size, and yet which describes the track's shape. Freeman¹⁵ has suggested that a useful approximation to curvature is the sequence of quantized directional segments generated by the points in a thinned track. Kuhl⁵

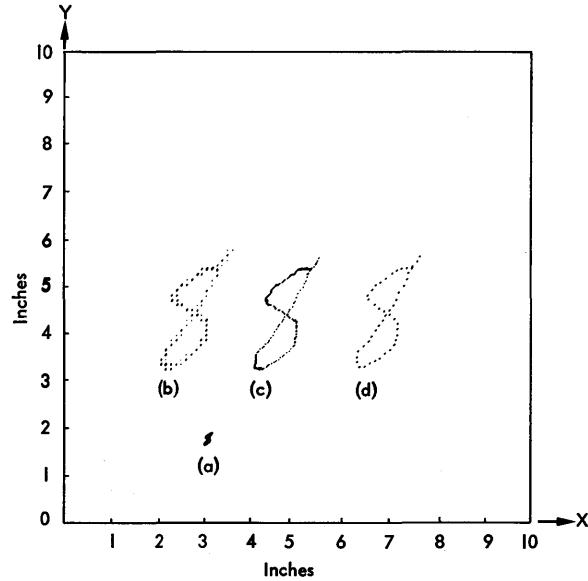


Figure 2. Display of a stroke—(a) as drawn, (b) magnified, (c) magnified and smoothed, (d) magnified, smoothed, and thinned.

and Bernstein¹⁰ have used this approximation in their character recognition schemes. Bernstein, in fact, found it unnecessary to use the duration of each quantized direction but, rather, simply listed *changes* in quantized direction. Whereas Kuhl and Bernstein both used eight possible directions, the recognition scheme described here uses only four. Four directions, used in conjunction with other features, provide sufficient description for recognition, yet result in fewer symbol variations than do eight directions.

When a new point (the j th) is accepted in the thinned track, a quantized direction is computed using these inequalities: (1) if $|X_{Tj} - X_{Tj-1}| \geq |Y_{Tj} - Y_{Tj-1}|$,

- (a) direction is right if $X_{Tj} - X_{Tj-1} \geq 0$
- (b) direction is left if $X_{Tj} - X_{Tj-1} < 0$

or (2) if $|X_{Tj} - X_{Tj-1}| < |Y_{Tj} - Y_{Tj-1}|$,

- (a) direction is up if $Y_{Tj} - Y_{Tj-1} \geq 0$
- (b) direction is down if $Y_{Tj} - Y_{Tj-1} < 0$

If the same direction occurs twice in succession and is not the same as the last direction listed in the sequence, then it is added to the list; otherwise it is discarded. This requirement causes further thinning.

Small hysteresis zones (about 16° wide) around the quantized direction zone borders prevent the quantized directions in the approximate track description from switching back and forth—say from

right, to up to right—when part of a track is drawn along one of these borders. If, for example, the pen is moving to the right, it must next proceed at an angle steeper than 53° in order for the direction to change from right to up.

Applying these rules to the thinned track of Fig. 2 results in the direction sequence left-down-right-down-left-up.

Inking and Corner Detection

The CRT displays the pen track as “ink” constructed of a sequence of directional segments originating at the pen-down location. Sixteen possible directions are required to provide the user with sufficient “ink” detail. A segment is added to the “ink” each time a new thinned data-point arrives.

This same sequence of directions is used to detect sharp corners. A corner is detected whenever the pen moves in the same (± 1) 16-direction for at least two segments, changes direction by at least 90° , and then proceeds along the new direction (± 1) for at least two segments. The change in direction must take place either immediately or through a one-segment turn. Applying this test to the track in Fig. 2 detects corners in the upper-left and lowermost parts of the stroke.

Size and Position Features

As a stroke is drawn, its x (horizontal) and y (vertical) extremes are continuously updated. When the pen is lifted, thereby indicating the completion of the stroke, the analysis scheme uses these extremes to calculate the symbol's height and width in fractions-of-an-inch, its aspect ratio (ratio of height to width), and its center relative to the tablet origin. It divides the rectangular area defined by the symbol extremes into a 4×4 grid. The starting (pen-down) and ending (pen-up) points, as well as the corner locations, are then each encoded as lying in one of these 16 areas, thereby locating them relative to the symbol. Figure 3 shows the 4×4 grid defined by the track in Fig. 2. This track has the following description:

1. Six 4-direction segments (left-down-right-down-left-up) encoded as 2-3-0-3-2-1.
2. Corners at positions 7 and 15.
3. Starting point at position 0.
4. Ending point at position 0.

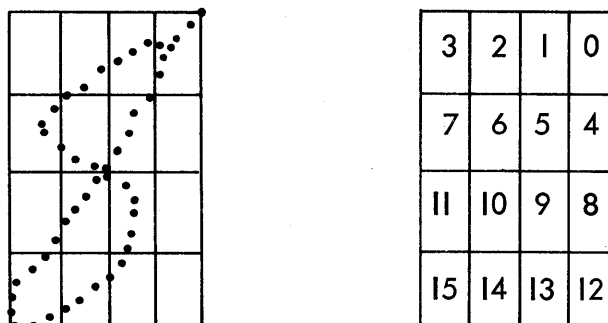


Figure 3. The 4×4 grid for the track of Fig. 2.

5. Height = 0.30 in.
6. Width = 0.22 in.
7. Aspect ratio = 1.36.
8. Center at $x = 3.15$ in., $y = 1.75$ in.

Computing Requirements

The data analysis described above takes up about 40% of the available computing time in the limiting case, where each data-point becomes a member of the thinned track. Smoothing and thinning requires about 20% of this analysis time, inking 30%, and corner detection 16%. The remainder of analysis time is for computing quantized directions, updating x and y extremes, and bookkeeping. The smoothing, thinning, and inking functions could be handled by hardware or by an I/O processor. Corner detecting could be deferred until stroke completion without requiring further storage (since it is based on the ink description, which is stored in any case), and without noticeably increasing the time between stroke completion and recognition. With these changes, a single user would require only about 15% of an IBM System/360 Model 40.

DECISION-MAKING

Several characteristics of the decision-making scheme should be pointed out before it is described in detail. When a stroke is completed, its description is added to those of the other strokes belonging to the same symbol. The decision-making scheme identifies the partial symbol corresponding to this set of strokes but does not display its identification at this time. Such a partial symbol is considered complete when the user pauses, draws a somewhat distant stroke, or draws a stroke which cannot be combined with the partial symbol to form one of the 53 allowable symbols. This symbol is then displayed. The decision-making scheme thereby separates symbols

from one another and recognizes them as they are written.

The identification of a symbol is based on a data-dependent sequence of tests. At each step in the decision-making process there are several potential identifications. Some of these are eliminated by testing one of the features of the track. The particular test applied at any step depends on the set of possible identifications at that step, and on those characteristics of the track which have already been examined. The decision-making scheme thus has a tree structure. Its original design was based on an examination of the handwriting of four users. The author changes its structure, to accommodate additional symbol variations, as he acquires more experience.

Identification of Single Strokes

The first test groups the single-stroke partial symbols according to shape. This is accomplished by locating a sequence of direction segments in a table.

Only a few segments should be looked up in the table because each additional segment increases the table length by a factor of four (recall that there are four quantized directions). However, if too few are looked up, the test will not sufficiently reduce the number of possible stroke identifications in each shape group. A good compromise is to look up the first four direction segments of a track. If a track has fewer than four direction segments, it is encoded such that the last segment is repeated until there are a total of four. The table is therefore 256 entries long—160 ($4 + 4 \cdot 3 + 4 \cdot 3^2 + 4 \cdot 3^3$) of which correspond to allowable encodings. For this test, the track of Fig. 2 is encoded as 2-3-0-3 (left-down-right-down). The possible stroke identifications corresponding to this table entry are S,5,8,9, and “cannot interpret.” Further tests, based on this par-

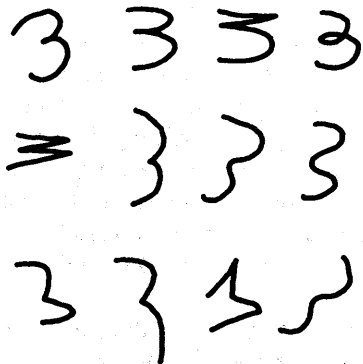


Figure 4. Some tracks recognized as the symbol 3.

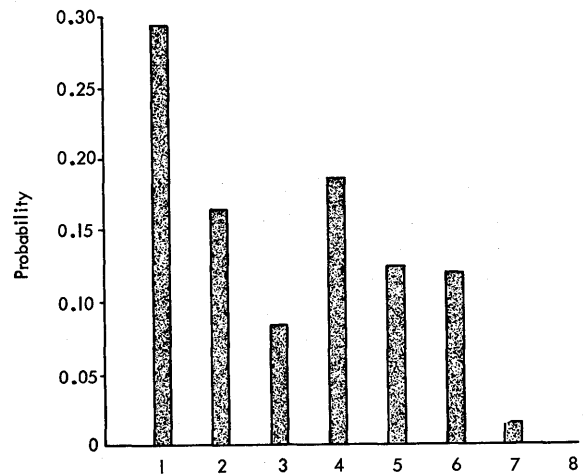


Figure 5. The probability of a number of symbols having the same first four direction segments.

ticular set of possibilities, result in a single identification.

The sequence of the first four direction segments of a track provides good first-level discrimination. In the present table, 45 of the 160 possible sequence encodings result in immediate identifications, with no further testing necessary. Another 50 sequences result in an immediate “cannot interpret.”

To allow for variations in handwriting, each allowable symbol has many table entries. A track for the symbol 3—which has one of the most complex and varied shapes—may be drawn having any one of 26 different sequences of the first four direction segments. Figure 4 shows some of the tracks identified as the symbol 3.

Assuming that each of the single stroke partial symbols is equally likely, and that each four-direction segment description of any particular symbol is equally likely, the probability of having a number of symbols with the same table exit can be calculated. Figure 5 shows this probability of requiring further testing to discriminate among a number of symbols upon exiting from the currently used table. Approximately 30% of the direction sequences require no further testing. The probability of requiring further testing to discriminate among as many as four, five, or six symbols is fairly high, because several symbols might have the same first four directions in common, but other features which are different.

After this first test, the stroke is either recognized or is known to the one of a particular set of two-to-seven symbols. The second test is one which best (in the author’s judgment) discriminates among these

particular symbols. Depending on the symbols, this test distinguishes differences in:

- The directions following the first four (e.g., to distinguish some 2's from some 3's)
- The number of corners (e.g., S vs 5);
- The positions of the starting or ending points relative to the symbol extremes (e.g., 0 vs 6);
- The aspect ratio (e.g., C vs ();
- The size (e.g., , vs));
- The position relative to a line of text (e.g., , vs ').

Testing continues until the number of possible stroke identifications is reduced to one. Figure 6 shows the sequence of testing required for the recognition of the track in Fig. 2. It is identified as the symbol 8.

The number of tests required for recognition is an interesting parameter because it measures the time and computer memory required for decision-making. The probability of requiring some number of tests for recognition can be calculated if it is assumed that each allowable single-stroke partial symbol is equally likely and that each decision tree-exit corresponding to a particular symbol is equally likely. Figure 7 shows the probability of requiring further testing after performing a number of tests in the existing decision tree. In more than half of all stroke recognition sequences, only two tests are required to arrive at a stroke identification. Figure 7 shows that ambi-

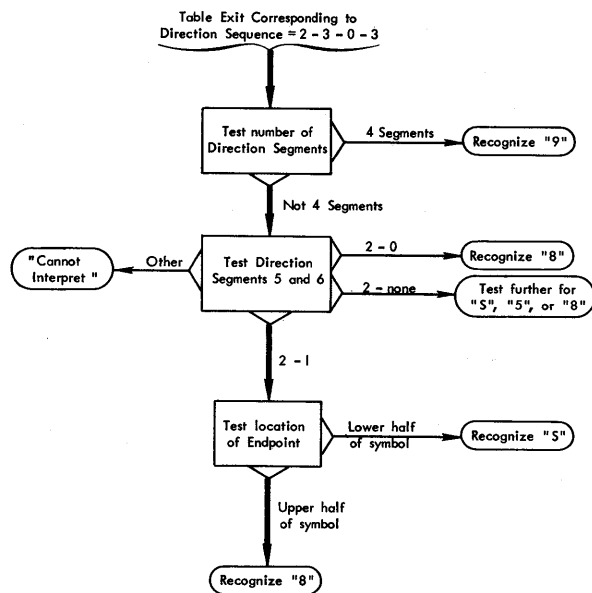


Figure 6. Recognition of the track of Fig. 2.

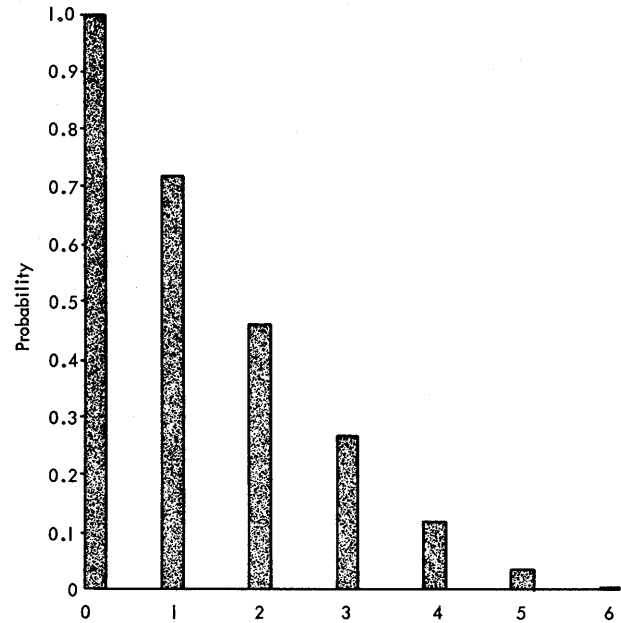


Figure 7. The probability of requiring further testing after n tests.

guity among several possible stroke identifications is always resolved after six tests.

Recognition of Multiple-Stroke Symbols

The recognition of a multiple-stroke symbol is based on the identifications and relative positions of its constituent strokes, but not on the order in which they are drawn. This is accomplished by cross-linking the tree structure. Each symbol has several descriptions, and therefore may be drawn in several ways. For example, a set of three horizontal strokes and one vertical stroke is recognized as the symbol E regardless of the writing order. The symbol E may also be written as an "L-like" stroke and two horizontal strokes, a "T-like" stroke and two horizontal strokes, a "T-like" stroke and one horizontal stroke, or a single "€-like" stroke.

In many situations, each stroke requires only a general, rather than a precise, identification. For example, if a stroke known to be part of a multiple-stroke symbol is drawn vertically downward ($\pm 45^\circ$), it need only be identified as a vertical stroke, rather than as a 1,), (, or /. This reduces the compounding of errors that would otherwise result from incorrect identifications of single strokes, and also simplifies the decision-making.

Just as the starting and ending points of a single-stroke symbol are encoded according to their positions relative to the x (horizontal) and y (vertical)

extremes of that stroke, the starting and ending points of each stroke in a multiple-stroke symbol are encoded according to their positions relative to the x and y extremes of the whole symbol. These encoded positions differentiate those symbols which are comprised of the same combination of strokes. At most, four symbols have the same set of strokes.

A symbol comprised of two strokes identified as verticals is a K, V, X, or Y. This symbol is recognized as a K if one of the strokes has both its starting point and ending point in the leftmost quarter of the symbol. It is recognized as a V if both of its ending points are in the middle part of the lower quarter of the symbol. It is a Y if it has one ending point which is neither in the leftmost quarter, nor in the lower quarter. Otherwise, this symbol is an X. Other ambiguities among multiple-stroke symbols are similarly resolved.

Segmentation into Symbols

As the user writes a line of text, the recognition scheme decides when a partial symbol is completed, recognizes and displays it, and begins the analysis of the next symbol as it is being written. The scheme separates the set of strokes making up a completed partial symbol from the first stroke in the next symbol by considering timing, and the geometric extents and identifications of the partial symbol and the following stroke.

If a prespecified time elapses following the end of the most recent stroke, the corresponding partial symbol is considered completed regardless of what follows. This between-symbol time delay must be greater than the maximum expected time delay between two strokes belonging to the same symbol—0.3 sec has proven sufficient for experienced users. A sufficient pause between symbols eliminates the need to test for geometrical separation. This procedure reduces normal writing speeds by a factor of about one-half, but results in the most accurate segmentation.

A partial symbol is considered completed, regardless of timing or position, if it cannot be combined with the following stroke to form an allowable symbol. The symbols 8, Q, A, and E are examples of partial symbols which cannot be combined with *any* other stroke to form an allowable symbol. If a partial symbol can be combined with some strokes, but not with others, then the following stroke is potentially identified on the basis of its first four quantized direction segments. A test is then made to determine if

the partial symbol can be combined with a stroke having this particular potential identification to form an allowable symbol. A partial symbol identified as a T, for example, may be combined with a vertical stroke to become an A, H, K, or *, or may be combined with a horizontal stroke to become an F or I, but cannot be combined with a potential C, U, or 2. In some situations, the following stroke must be identified more precisely. A circular stroke followed by a normal size “/-like” stroke, for example, may be identified as the letter O; but a circular stroke followed by a short “/-like” stroke will be identified as the number 0 followed by a comma.

Strokes which are written in quick succession, and which can be combined to form an allowable symbol are tested for geometric separation. Two such schemes for separating symbols geometrically have been investigated at RAND. One of these assumes that the writing surface is an 85×42 array of symbol spaces, and that each symbol is drawn in a different space. The other scheme assumes that the strokes in a symbol overlap (or, in some cases, are close to one another), but do not overlap with strokes belonging to another symbol.

Fixed Symbol Spaces. This scheme tests to see if the centers of a partial symbol and that of the next stroke lie in the same symbol space. If they do not, it separates them unless the partial symbol is a single vertical stroke lying just to the left of the following stroke. If they do lie in the same space, it considers them as part of the same symbol unless the stroke is a vertical in the rightmost quarter of the symbol space, *and* the partial symbol cannot be combined with a *right-hand* vertical to form an allowable symbol. These adjustments on the basic test allow for some misplacement of vertical strokes.

The user must be made aware of the symbol space positions so that he may space his printing accordingly. This is accomplished by displaying a series of dimly lit vertical bars, each at a border of a symbol space.

The primary disadvantage of this scheme is that small displacements of strokes relative to the symbol spaces cause segmentation errors. Such displacements may occur when the user writes quickly, or when the hardware displaces the “ink” slightly, thereby misinforming the user as to where he is writing. A further disadvantage is that the vertical bars are distracting.

Relative Symbol Positions. This scheme tests for the overlapping or adjacency of the partial symbol and

the following stroke. If the horizontal-extent of the stroke lies within the horizontal-extent of the partial symbol, it considers them parts of the same symbol. Partial-overlap causes the partial symbol and the following stroke to belong to the same symbol, unless their centers are farther apart than a maximum allowable distance. No-overlap separates the partial symbol and the following stroke, unless one (or both) is a single vertical stroke and their centers are not separated by more than a maximum allowable distance. The allowable separation distances are based on the normally expected symbol size.

The maximum-distance-between-centers test enables the separation of two symbols, such as T/ or R2, which happen to overlap because they are written close together. When symbols having a vertical stroke are written, the vertical frequently does not overlap the remainder of the symbol. The position test in the no-overlap case takes this into consideration. It is particularly useful when | | will be made an H or N by the next stroke, and also when two parts of the same symbol—e.g., the 1 and 3 of B—are just slightly disconnected.

The disadvantages of this scheme are that it may separate two parts of the same symbol which are not written close enough, and that it may not separate two different symbols which are written such that they overlap slightly. Its advantage is that it works about as well as the fixed-symbol-space scheme, yet does not *require* the display of symbol space separators. It uses the displayed symbols to cue the user about how large to write and about how far apart to place symbols. Since the symbols are displayed with fixed spacing, users tend to inadvertently imbed blank spaces unless they are provided with some strong indication of the location of the symbol spaces. This scheme therefore works best when the vertical bars are displayed.

EVALUATION

One measure of the value of the recognition scheme is the accuracy with which it recognizes individual symbols. Since the goal of the scheme is to facilitate man-computer communication, a more direct measure of its value is the time required to perform some writing task on-line.

Three groups of users participated in the following two experiments. The first group consisted of six users who have had considerable experience with the symbol recognition scheme. The design of the decision-making scheme was, in fact, based on the

handwriting of four of these users. The second group consisted of seven engineers and programmers who have had experience using the RAND tablet, but have not previously written with the symbol recognition scheme. The third group was made up of nine engineers, programmers, and secretaries who have had no previous experience with either the tablet or the recognition scheme.

Accuracy of Recognition

Each user in the second and third groups participated in a half-hour training session prior to the testing phase of this experiment. The session began with an explanation of how to use the tablet, what symbols are recognized, the use of the scrub, etc. The user was then allowed to write whatever he wanted for 20 minutes—most users printed words, phrases, or the alphabet, usually repeating a symbol until it was recognized correctly. During the next 10 minutes, the user attempted to print each of the symbols while the author pointed out how he might achieve more success. The users in the first group did not receive any special training, since they were already familiar with the tablet and recognition scheme.

Since this experiment was concerned with *isolated* symbols, the decision-making scheme was adjusted (during the testing phase) to separate symbols on the basis of time only. Each user was instructed to carefully print the list of symbols five times, waiting for each symbol to be recognized before printing another. (The apostrophe was not included because the users were given no position cues, and therefore could not be expected to write an apostrophe different from a comma.) Each user thus printed each of 52 symbols five times; 53 responses were possible (including "cannot interpret"). All errors (including user errors such as writing 0 rather than Ø for the letter O) were recorded.

The average recognition rate for the group with experience in using the symbol recognition scheme was 93% correct. The lowest "score" in this group was 90%; the highest, 96%. The average score among those experienced previously with only the RAND tablet was 88%, with a low score of 82% and a high score of 92%. The average score among those with no previous experience whatsoever was 87%, with a low of 81% and a high of 93%. The recognition rates were not only high, but quite uniform. In fact, the "scores" for users in the second and third groups are indistinguishable. The time it

takes to learn to use the hardware—with its separate tablet writing surface and CRT working surface—therefore appears to be less than one-half hour. The users in the first group probably scored somewhat higher than the others because they were more familiar with the scheme, and because it was designed to accommodate their writing styles.

Many of the errors were due to the misrecognition of (,), [, and]—(, for example, was frequently confused with 1, C, or L. The asterisk was also frequently missed because its shape is not well defined, and it therefore has many variations. Some users not familiar with coding form conventions found it difficult to learn to write the letter O with a slash to distinguish it from the number 0, yet learned to print Z with a crossbar, and I with upper and lower bars. The remainder of the errors were due to confusions between 7 and >, G and C, 5 and S, + and T, and a wide variety of other pairs of symbols.

Ease of Operation

Since this recognition scheme is designed for use in a problem-solving environment, an experiment was performed in which programmers used the scheme while solving simple coding problems. A user was given flow charts (Fig. 8), and was asked to write the corresponding computer code for one problem directly on-line (using only the recognition scheme) and that for the other off-line (using pencil and paper). The times required for solution—a measure of the usefulness of the recognition scheme versus pencil and paper—were recorded.

Since time was important, this experiment used the decision-making scheme which separates symbols according to identity and relative position as well as time. Users could therefore print quickly without pausing between symbols. Vertical bars were displayed to indicate symbol spacing. Users were allowed to make use of all editing facilities—i.e., change a symbol by writing over it; erase a symbol, space, or line with a scrubbing action; use the symbol \wedge to insert symbols between others already printed on a line; and use the symbol $>$ to obtain a blank line between two printed lines of text.

Eight programmers participated in this experiment. Four from the first group (described above) coded the problems in IBM System/360 Assembly Language. Each problem required the printing of approximately 200 symbols in this language. Two programmers each from the second and third groups coded the problems in FORTRAN—this required

Table 1. Times to Code the Problems of Figure 8

Group	Problem	Coding Form Time	On-Line Time
1	A	8¾ min	12 min
	B	9 "	9½ "
2 & 3	A	7 "	12½ "
	B	4 "	9 "

the printing of about 150 symbols for each problem. Half of the users in each group coded Problem A directly onto a coding form and Problem B directly on-line, using the tablet hardware and recognition scheme; the others coded Problem B onto a coding form and Problem A on-line.

Table 1 summarizes the results of this experiment. Each entry is an average for two users. The inexperienced (second and third group) users spent a large part of their on-line problem-solving time trying to communicate with the recognition scheme. The experienced users, who had harder problems (assembly language versus FORTRAN) to solve, were not, however, slowed down very much by the recognition scheme.

The subjective findings of this experiment are, perhaps, more important than the numerical results. The accuracy of recognition here was much lower than in the previous experiment, partly because the scheme had to separate symbols geometrically, but also because the inexperienced users had forgotten some of what they had learned about the scheme, and because all of the users were less careful. All users found it difficult to print symbols as small and as closely packed as the hardware generated them—85 across a 10-in width. They tended to print oversize symbols, thus causing the scheme to introduce blank spaces or to recognize a single multiple-stroke symbol as two symbols.

The editing facilities helped to compensate for the difficulties with symbol recognition. The users found it convenient to change a symbol by writing directly over it, or to erase an entire line with a single scrubbing action—editing procedures which are much more difficult to do with pencil and paper than with the tablet hardware. The symbol \wedge was frequently used to insert one or more symbols between two others. Although the symbol $>$ was used on only two occasions to insert a new line between two others, this editing feature is potentially valuable for composing solutions to more difficult problems.

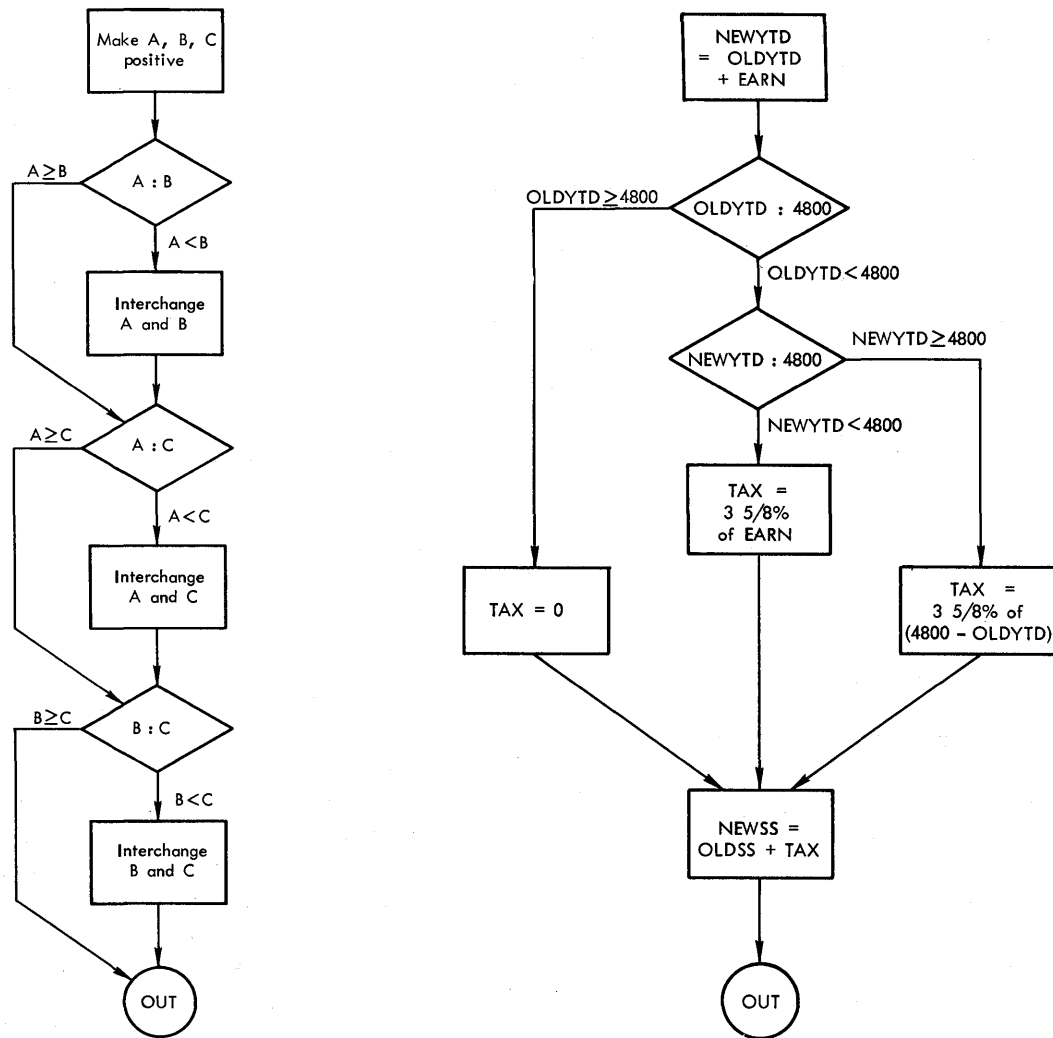


Figure 8. Flow charts of the problems coded in the experiment—(a) Problem A: a procedure for computing social security tax; (b) Problem B: a method of sorting three numbers into a descending sequence of their magnitudes.

Thus, this on-line recognition scheme with editing facilities appears to be a useful problem-solving aid, particularly as the users become more experienced, and the problems become more difficult. In a problem-solving situation, the editing facilities give the user much greater flexibility than pencil and paper. The accuracy of the recognition scheme is high enough that it is not distracting and the hardware is natural to use.

DISCUSSION

This recognition scheme meets its primary objective of enabling any user to communicate naturally

with a computer. A user is not distracted by any operational mechanics but, rather, may concentrate on his problem. All communications are made with a single device—a pen-like instrument. A user may write anywhere on a horizontal writing surface. The recognition scheme has been extended to recognize such flow-charting symbols as rectangles, circles, triangles, and diamonds. Thus, the user may draw free-form flowcharts as well as text, using only a pen.

The recognition program responds quickly and is efficient in storage. When the time-delay normally used to separate symbols is set to zero, the lifting of the pen and the display of a recognized symbol are apparently simultaneous. The recognition program—

including the data analysis and decision-making routines and data storage but not display or editing routines—requires about 2400 32-bit words of memory.

The major shortcoming of the present scheme is its difficulty in recognizing quickly written text: it has difficulty in separating overlapping symbols and in combining disconnected parts of a single symbol. Furthermore, since quickly written symbols tend to be distorted, they are misrecognized more often than carefully drawn symbols. These problems are due, in part, to the small size and close packing of the symbols. They can be relieved by allowing larger between-symbol spaces.

Another difficulty with the recognition scheme is that only a person familiar with the computer program can add a new symbol or new symbol description. Such changes are complicated because they frequently require several coding changes in the cross-linked tree structure of the decision-making scheme. A useful variation of this scheme would therefore be based on a data-dependent sequence of tests, but would permit automatic changes.

Finally, the advantages and disadvantages of pen-location-by-pen-location feature extraction should be clarified. This procedure is useful for real-time recognition because it minimizes the time delay between symbol completion and identification, yet produces a valuable set of features. It may, however, result in symbol variations not introduced by a scheme which extracts features after a symbol is completed. Such variations arise because some symbols may be drawn either clockwise or counterclockwise, portions of some symbols may or may not be retraced, some symbols may be constructed of various numbers and sequences of strokes, etc. Perhaps some other set of dynamically extracted features would prove as useful for discrimination as the present set without introducing as many variations.

ACKNOWLEDGMENTS

I would like to thank T. O. Ellis, J. F. Heafner, and W. L. Sibley, all of The RAND Corporation, for many profitable discussions, and for designing and implementing the graphical hardware-software without which the recognition scheme could not exist.

REFERENCES

1. M. E. Stevens, *Automatic Character Recognition—A State-of-the-Art Report*, National Bureau of Standards NBS Tech. Note 112 (May 1961).

2. E. E. David, Jr., and O. B. Selfridge, "Eyes and Ears for Computers," *Proc. IRE*, vol. 50, no. 5, pp. 1093–1101 (May 1962).

3. E. E. Graziano, *Automatic Pattern Recognition During the Period 1961–1962: An Annotated Bibliography*, Lockheed Missiles and Space Company, Sunnyvale, Calif. Report 6-90-63-16/SB-63-13 (May 1963).

4. E. C. Greanias, et al, "The Recognition of Handwritten Numerals by Contour Analysis," *IBM J. of Res. and Dev.*, vol. 7, no. 1, pp. 14–21 (Jan. 1963).

5. F. Kuhl, "Classification and Recognition of Hand-Printed Characters," *IEEE International Convention Record*, 1963, part 4, pp. 75–93.

6. T. Marill, et al, "Cyclops-1: A Second-Generation Recognition System," *AFIPS Conference Proceedings (FJCC)*, vol. 24, Spartan Books, Baltimore, 1963, pp. 27–33.

7. H. A. Glucksman, "A Parapropagation Pattern Classifier," *IEEE Trans. on Elec. Computers*, vol. EC-14, no. 3, pp. 434–43 (June 1965).

8. T. L. Dimond, "Devices for Reading Handwritten Characters," *Proc. Eastern Joint Computer Conference*, Dec. 1957, pp. 232–37.

9. L. D. Harmon, "Handwriting Reader Recognizes Whole Words," *Electronics*, vol. 35, no. 34, pp. 29–31 (Aug. 24, 1962).

10. M. I. Bernstein, *Computer Recognition of On-Line, Hand-Written Characters*, The RAND Corporation, Santa Monica, Calif., RM-3753-ARPA (Oct. 1964).

11. R. M. Brown, "On-Line Computer Recognition of Handprinted Characters," *IEEE Trans. on Elec. Computers*, vol. EC-13, no. 6, pp. 750–52 (Dec. 1964).

12. W. Teitelman, "Real-Time Recognition of Hand-Drawn Characters," *AFIPS Conference Proceedings (FJCC)*, vol. 26, part 1, Spartan Books, Baltimore, 1964, pp. 559–75.

13. M. I. Bernstein, *An On-Line System for Utilizing Hand-Printed Input*, System Development Corporation, Santa Monica, Calif., TM-3052, July 11, 1966.

14. M. R. Davis and T. O. Ellis, "The RAND Tablet: A Man-Machine Graphical Communication Device," *AFIPS Conference Proceedings (FJCC)*, vol. 26, part 1, Spartan Books, Baltimore, 1964, pp. 325–31; also, The RAND Corporation, Santa Monica, California, RM-4122-ARPA (Aug. 1964).

15. H. Freeman, "On the Encoding of Arbitrary Geometric Configurations," *IRE Trans. on Elec. Computers*, vol. EC-10, no. 2, pp. 260–68 (June 1961).

BASIC HYTRAN SIMULATION LANGUAGE—BHSL

Jon C. Strauss *

*Electronic Associates Inc.
Princeton, New Jersey*

INTRODUCTION

An appropriate subtitle for this paper might read: "A Fortran Compatible Dialect of the SCI Continuous System Simulation Language." Here the words "continuous system" † distinguish the application area of interest from that encompassed by the event based simulation languages of the genre of GPSS, SIMSCRIPT, etc. The reference to SCI indicates that BHSL, while certainly a member of the growing family of simulation languages,^{1,2} was designed to meet standardization guidelines established by the Simulation Software Committee of Simulation Councils Incorporated.³

The Basic Hytran Simulation Language and the associated translator and runtime system are a programming system for the EAI 8400 digital computer. It serves as a basis for the complete Hytran Simulation System on the EAI 8900 Hybrid Computing System.

The primary aim of BHSL is to provide a problem oriented vehicle for the representation (description) of continuous dynamic systems that can be modelled by sets of ordinary differential and/or difference

equations in one or more independent variables. The language includes a set of control statements which permit the simulation analyst (programmer) to exercise control over the solution of the equations representing his problem. This control can be programmed into a simulation program or introduced at execution time by use of an associated interactive command and control language system.⁴

Design Objectives

The Continuous System Simulation Language (CSSL) was designed to incorporate the better features of the previous dynamic system simulation languages. These features include:

1. Automatic sequencing of operations to optimum calculational order,
2. Problem oriented operators and diagnostics,
3. Mnemonic variable and operator naming, and
4. Expression and statement constructions.

The design process was guided by several broad objectives on CSSL programs. These guidelines indicated that CSSL and CSSL programs should be:

1. Easily adapted to various levels of programmer and problem sophistication,
2. Modular (include user written problem

* Presently with Carnegie Institute of Technology, Pittsburgh, Pa.

† Systems in which the response phenomena occur continuously in one or more independent variables as opposed to discrete systems in which the response phenomenon occurs as a sequence of events at discrete, possibly random, points in an independent variable (nominally time).

- oriented operators in the CSSL vocabulary),
3. Conversational (provide for exception only, conversational interaction with a running CSSL program), and
 4. Efficient (contain a variety of user selected and controlled numerical methods).

These considerations are discussed in detail in Reference 3.

BHSL includes all the design requirements of CSSL, but the dependence on a general procedural language has been particularized to Fortran IV. Fortran statements are used directly for sophisticated procedural operations (i.e., formatted I/O, conditional logic, and algebraic processing). In addition, the BHSL processor translates the source program to a valid Fortran IV program for subsequent compilation by the EAI Fortran IV compiler. This technique not only simplifies the implementation task, but it provides for more efficient object code than would be likely in an initial compiler implementation.

Language Features

BHSL can be used as, and contains the desirable features of, the block diagram description languages (e.g. SCADS, MIDAS, etc.). At the next level of problem description sophistication, an equation based notation similar to that of Fortran can be used as in MIMIC and DSL/90. For more sophisticated problems (and programmers), a BHSL program can be expanded to include all the capabilities of Fortran, still retaining the inherent language features of integration, special simulation oriented operators, optimum sequencing of program operations, and problem oriented diagnostic checking. This feature is termed programmable structure.

As a trivial example of the flexibility of the representation aspects of the language, one might choose to describe the differential equation

$$\frac{d^2x}{dt^2} + a \frac{dx}{dt} + b x = f(t)$$

$$\left. \frac{dx}{dt} \right|_{t=t_0} = DX0$$

$$\left. x \right|_{t=t_0} = X0$$

in a differential equation oriented form:

$$\begin{aligned} DX &= \text{INTEG} [DX0, F - B*X - A*DX] \\ X &= \text{INTEG} [X0, DX] \end{aligned}$$

or in an analog computer equation oriented form:

$$\begin{aligned} S &= -(-F + 10*POT1 + 10*POT2) \\ DXM &= -\text{INTEG} [-DX0, S] \\ X &= -\text{INTEG} [X0, DXM] \\ DX &= -DXM \\ POT1 &= (A*DX) / 10 \\ POT2 &= (B*X) / 10 \end{aligned}$$

or one might add new operators to BHSL and describe the problem exactly as though programming an analog computer:

$$\begin{aligned} A1 &= \text{AINT} [P3, F, P1, P2] \\ A2 &= \text{AINT} [P4, A1] \\ A3 &= \text{INV} [A2] \\ P1 &= \text{POT} [A, A1] \\ P2 &= \text{POT} [B, A3] \\ P3 &= \text{POT} [DX0, -10] \\ P4 &= \text{POT} [X0, 10] \end{aligned}$$

The most important user feature is the macro. The macro consists of a named set of prototype statements that are inserted into the program each time the name is referenced. During insertion, certain of the variable names are altered in order to maintain the requisite uniqueness properties. The macro is inserted into the source program before the statements are sorted to optimum calculational order. This preserves the true parallel system description aspect of the language.

The simulation oriented BHSL operators, with the exception of the integrator, are mechanized as system macros. The user may also define macros for individual problems to represent commonly used sets of descriptive information. These macros are referenced in his program exactly as though they were operators of the language. In addition, the user has the capability of creating his own macro library which will appear to the system as an extension and/or replacement of the system library. Thus, each user may completely alter the semantics of the language while preserving the syntax and the structure of the operational environment.

The macros defining the problem oriented operators used in the preceding example could be defined as follows:

```

≡ ANALOG INTEGRATOR ≡
MACRO [AINT [OV = IC,A1,B1,C1, **
A10,B10,C10]]
  STANDVAL [IC,A1,B1,C1,A10,B10, **
C10/0,0,0,0,0,0]
  OV = -INTEG [IC,A1 + B1 + C1 **
+ 10*(A10+B10 + C10)]
ENDM

≡ POTENTIOMETER ≡
MACRO [POT [OV = A, IV]]
  OV = (A*IV)/10
ENDM

```

System Organization

The language is designed to augment the capabilities of Fortran through the addition of certain problem oriented simulation operators (e.g., integrator), problem oriented syntax (e.g., user defined macros, free format input, etc.), and implicit organizational features (e.g., sorting of statements into optimum calculational order). The translator processes a simulation program which consists of a collection of BHSL statements and Fortran statements by translating to a valid Fortran intermediate program (target program). This program has the requisite structure for interfacing with the runtime system. The monitor calls in Fortran to compile the target program and, under user directive, initiates the execution of the compiled simulation program.

FUNCTIONAL DESCRIPTION OF BHSL ENVIRONMENT

The presentation of a problem oriented language is facilitated by a general discussion of the application area for which it is designed.

Figure 1 presents a block diagram of the calculational flow of a general digital simulation program that involves integration of ordinary differential equations in a single independent variable. The various main regions of the program have been denoted as the Initial Region, the Dynamic Region, and the Terminal Region. It is convenient for descriptive purposes to refer to a single run of a simulation program (i.e., solution of the differential equations over the desired independent variable range) as a *case*. A set of consecutive runs of the same program (with altered parameters or I/O) is referred to as a *job*.

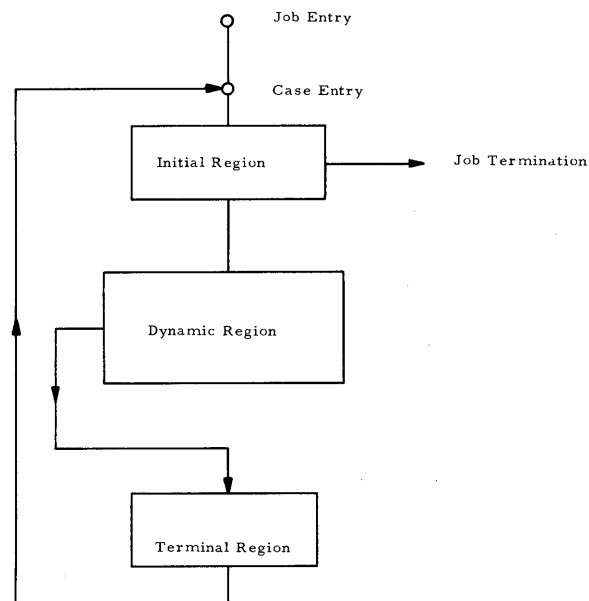


Figure 1. Overall structure of a digital simulation program.

Initial Region

The initial region encompasses all those calculations, input/output operations, and initializing procedures that must be performed prior to one case of a series of calculations (integration) at discrete points on the independent variable. Initializing operations of a more permanent nature (e.g., read in of a particular integration algorithm) would be performed prior to entering this region.

Figure 2 indicates that the initial region contains three types of operations: interpretive input/output, general initial calculations, and integration initialization.

The Interpreter routine facilitates run time interaction between the simulation analyst and the program. The analyst enters parameter and system initialization commands from a console. These commands, which are by exception only, are translated and executed at run time. The following commands would be helpful in this problem environment.

1. Adjust any name variable in the simulation (e.g., a parameter or initial condition).
2. Perform immediate readout of the value of any addressable variable or parameter in the simulation.
3. Exercise control over all the adjustable parameters of the integration algorithm which would include interval, minimum

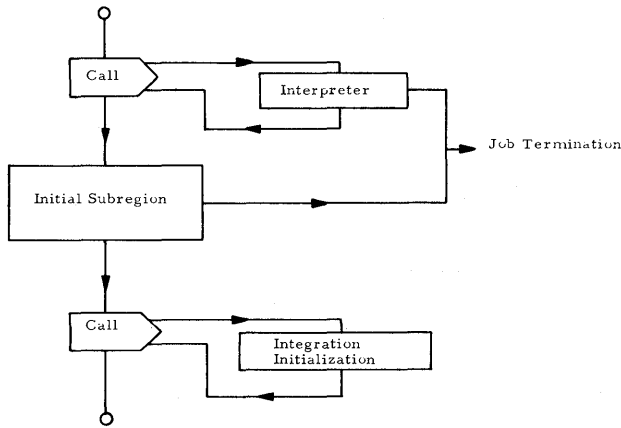


Figure 2. Structure of the initial region.

interval, initial and final values of the independent variable, and error control.

4. Perform simple arithmetic calculations at the console for computing changes in problem parameters on the basis of past results.
5. Provide initiation control of individual simulation runs (cases) and termination control of a set of runs (job).

Dynamic Region

The dynamic region is that portion of the simulation which takes an active part in the interaction between the digital computer and the external world. It represents all the calculations and I/O operations performed at each user defined discrete value of the independent variable.

The basic interval in the independent variable represented by each pass through the dynamic region is termed the *communication interval*. This interval is determined solely by the accuracy requirements on the communication with the external world. The interval at which various portions of the calculation (integration) are being updated is generally smaller than the communication interval. This *calculation interval* is determined strictly by the accuracy requirements on the digital calculation (especially the integration).

Figure 3 illustrates that the dynamic region can be described functionally in terms of input/output and integration.

The input/output subregion constitutes those actions performed in the basic independent variable loop other than the integration calculations (if any).

It mechanizes any time dependent algebraic calculations that are not an integral part of the derivative calculation and also includes all digital input necessary in the dynamic loop and testing of program conditions. These tests might determine whether to: 1) terminate the job or case by transferring control to the terminal region; 2) terminate the case by transferring control to the initial region; or 3) calculate new history information and restart the integration. All output of system variables at the communication rate is mechanized from this subregion.

The Integration subregion includes all integration being performed with respect to the independent variable of the dynamic region. To provide for different integration rates between sets of state variables of a simulation, this may be structured by the programmer to allow an arbitrary number of *sections*. These sections are generally calls to a system integration routine which integrates a portion of the state vector over a specified interval in the independent variable. Certain of the sections, however, might not involve integration at all. These could be programs simulating portions of parallel synchronous logic which need to be clocked at a different rate than that of the integration.

The specified interval associated with a section is the section communication interval. It would be set larger than the system communication interval for an integration section being updated at a slower rate than the system communication frequency and smaller in those cases where it was necessary to have communication between the variables of differ-

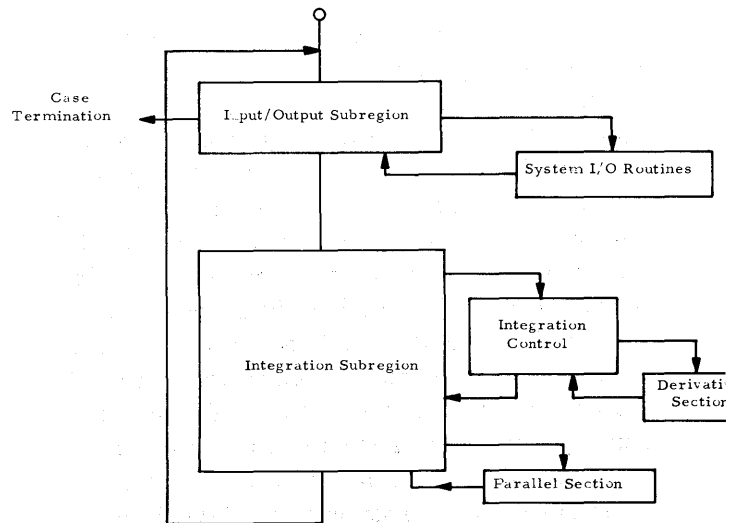


Figure 3. Structure of a dynamic region.

ent integration sections at a higher rate than that of communication. (This would likely be the case when simulating parallel logic.) The various sections are separated by procedural (Fortran) coding for both determining the frequency at which the integration sections are entered and possibly performing simple interpolation on those state variables being updated at a slower rate.

Each section involving integration has associated with it a subprogram for calculating the derivatives of the state variables being integrated. Since there is one such subprogram for each integration section, the term *derivative section* is used in the description that follows to indicate a section involving integration and its associated derivative subroutine. The system integration package and the various derivative sections share a common symbol table so that there can be direct communication between the various variables and derivatives.

Depending on the integration algorithms employed, there are various combinations of step size, interval alteration and corrector iteration algorithms, etc. that must be specified for each derivative section to assure accurate numerical integration.

Terminal Region

Figure 1 shows that the terminal region receives control from the dynamic region and returns control to the IC (Case) entry. The terminal region contains calculations and I/O necessary to properly terminate a single case. In addition, some system bookkeeping operations such as plot output are performed at this point in the simulation.

DESCRIPTION OF BHSL

Not surprisingly, the gross structure of a BHSL Program and its operating environment bear a marked similarity to the general structure just outlined. The following discussion delineates the specifications for a source program, the resulting object program, and the runtime system.

BHSL Statements

A program is written as a sequence of statements structured (either explicitly or implicitly) into functional groupings termed blocks. The action statements of a block are either representation statements or procedural statements. The statements that delineate the range of a block are structure statements and the statements that indicate how the block is to be

processed (both for translation and execution) are control statements.

Although the syntax of each statement type is basically different, the format of all statements on the physical records of the input media is identical. Statements may be started at any position of the physical record and are continued across physical records with an explicit continuator character (###). Either an end of record or an explicit terminator character (;) serves to terminate a statement.

Representation Statements describe the physical (mathematical) system to be simulated (solved). These statements are the heart of the simulation language; they are similar to the assignment statements of Fortran.

Each statement defines (determines the values of) one or more unique output variables as the result of one or more operators operating on a set of input variables. The variables may be either of type real or logical. In addition to the conventional arithmetic, logical, and relational operators of Fortran, a user expandable set of simulation oriented operators is included in the language system.

For example, in the representation statement:

```
X = INTEG [X0, A*X1 + COS (X2)]
```

X is the output variable, X0, A, X1, and X2 are the input variables, and INTEG is a simulation operator.

Procedural Statements are standard Fortran statements that have been couched in the format of BHSL. They are separated physically from the representation statements by block groupings and the translator only performs text editing functions.

For example, the procedural block acts externally as a representation statement, but its actions are defined by standard Fortran statements. The following block represents a limiter with the defining equations:

$$y = \begin{cases} a & \text{for } x \leq a \\ x & \text{for } a < x < b \\ b & \text{for } x \geq b \end{cases}$$

```
PROCEDURAL [Y = A,X,B]
```

```
IF (X.LT. A) Y = A
```

```
IF ((A.LE. X) .AND. (X.LE. B)) Y = X
```

```
IF (X.GT. B) Y = B
```

```
END
```

The above collection of statements is sorted collectively as a single representation statement with output variable Y and input variables A,X, and B.

Structure Statements delineate explicit structural groupings of the other statement types. These groupings are treated in a different fashion by the translator according to type. The first statement of the preceding example is a structure statement.

Control Statements are utilized to instruct the translator and/or the execution monitor as to how the program should be processed. Had it been desired to use conditional transfers to symbolic labels in the above procedural block, the programmer would declare the labels to be used with a LABEL control statement as follows:

```
PROCEDURAL [Y = A,X,B]
  LABEL [L1,L2,L3]
  IF ((X .LT.A) .OR. (X .GT.B)) GO ##
    TO L1
  Y = X; GO TO L3; L1: IF (X .LT.A) ##
    GO TO L2;
  Y = B; GO TO L3; L2: Y=A;
  L3: CONTINUE;
END
```

Blocks

Blocks are initiated by an appropriate structure statement and terminated with the END statement.

Regions are the highest level BHSL blocks. The initial region is a set of procedural and control statements that act as the functional equivalent of the initial subregion of Fig. 2. The other functions indicated in Fig. 2 are provided automatically by the translator.

The terminal region also contains only procedural and control statements; it is the functional equivalent of the terminal region in Fig. 1.

The dynamic region is the functional equivalent of Fig. 3 and may contain any of the statement types of BHSL. In general, the statements of this region are structured into derivative and parallel sections separated by procedural and control statements.

Sections contain representation statements and pseudo sections (which act like representation statements, e.g., the procedural block presented above). Certain control statements which specify the error control procedures for the execution monitor are also permitted. Prior to the specification of CSSL,³ digital simulation languages, in general, constituted little more than the derivative section of BHSL.

The statements of a section are sorted by the

translator to optimal calculational order prior to inclusion in the target program. In general, the sort algorithm requires all the input variables of a statement to have been defined before the statement can be processed (and so define the output variable). Sort loops are broken implicitly by memory type operators such as integrators and time delays and explicitly by certain algebraic iteration operators.

All integration operators are mechanized by the centralized integration system which is contained in the runtime system. The derivative section is translated to a Fortran subroutine which calculates the state variable derivatives. The execution of the subroutine is controlled by the integration routines. The parallel sections are also translated to subroutines, but the control flow is inserted directly into the program. Parallel sections contain no integrate operators; they are generally used to represent a collection of parallel logic elements which must be clocked at a different rate than the integration.

Source Program

A program includes all the statements and/or blocks necessary for both the representation of the simulated system and the control of the simulation itself. A BHSL program has associated with it a set of unique output variables and a unique independent variable. The body of a program may be structured into three types of regions. If no explicit structuring is indicated, the translator assumes that the whole program represents a single derivative section and it inserts a standard central program.

The BHSL translator operates on the source program to produce Fortran IV then compiled by the Fortran IV compiler to yield an object program which interfaces properly with the runtime system.

Object Program

Certain elements of the CSSL do not appear explicitly in BHSL. These features, which include segmentation, multiple independent variables, etc., can be achieved by making Fortran patches in the target program.

Runtime System

In addition to the standard monitor functions (i.e., subroutine loading, priority interrupt scheduling, etc.), the simulation system requires two spe-

cial execution time routines. These are the integration system and the I/O control system.

The *integration system* has two entries, one for initialization and one for integration. In addition to setting up the initial conditions on the state variables of the integration, the initialization entry also allocates memory for the history information required by the particular integration algorithms. The integration entry transfers control to the appropriate algorithm to integrate the specified derivative section over its communication interval.

The system includes a variety of algorithms and error control options which may all be altered at execution time under interpretive control. The integration algorithms include Euler, Runge-Kutta, Third, Fifth and Seventh Order Adams Predictors and Adams Predictor Correctors. Where applicable, error control options include adaptive quadrature error control on individual state variables through iteration and interval alteration.

The *I/O control system* is a collection of input/output routines which includes the interpreter, print plot, and general output formatting routines.

This interpreter is an upwards compatible version of the Hytran Operations Interpreter.⁵ In its most simple form, it contains algebraic capabilities and is able to respond to combinations of appropriate BHSL control statements. The interpreter provides readout and data alteration by exception only.

Interpretive Command and Control Features

The system is designed such that a program may be executed under control of a set of interpretive instructions. In any single program, this set might be only a simple initiate execution command or it might include a string of control statements specifying alterations to the program and its data. The programmer specifies the point, if any, in his program at which he wishes to accept interpretive command and control information with the INTERPRETER control statement.

EXAMPLES

The examples of this section are concerned with the nonlinear two point boundary value problem:

$$\begin{aligned} \ddot{y} &= -(1 + e^y) \\ y(0) &= 0, y(t_f) = 1.0 \end{aligned}$$

This problem was solved as an example in a recent article describing the MIDAS III simulation lan-

guage.⁵ The reader will no doubt find it interesting to compare the means used to represent the problem and control the solution in the two languages.

The object of the problem is to determine the unknown initial condition on y such that the terminal condition is met. This constitutes a solution; given two initial conditions, it is a trivial matter to integrate the equation over the independent variable range to determine the trajectory $y(t)$. Questions of existence and uniqueness of solutions of boundary value problems, while certainly important, are ignored in this discussion.

Implicit Structure Program

The program presented below is designed to be used in an interactive fashion for the solution of the boundary value problem. In the sample interactive dialog, an analyst exercises this program from a digital I/O station in much the same fashion that he would interact with an analog computer.

```

IMPL
  ≡ IMPLICIT STRUCTURE EXAMPLE ≡
  ≡ HAND OPTIMIZED SOLUTION ≡
  INTAL [RK4]; COMDEL [0.1]; CALF [10];
  TERMVAL [1.0]
  TITLE [TWO POINT BOUNDARY VALUE #
  PROBLEM—J. C. STRAUSS]
  DATA [Y0,DY0/0,1]
  Y = -INTEG [Y0, INTEG [DY0, 1 + EXP
  [Y]]]
  SAVE [Y,T]
END

```

The first action taken in an implicit structure program is to transfer control to the interpreter. Since an initial guess for $y(\emptyset)$ is supplied to the program (via a DATA statement), the analyst responds to interpreter's request for input with:

```
GO;
```

This command causes control to be returned to the program which computes a complete solution for y on the interval $0 \leq t \leq 1$.

The TERMVAL [1.0] control statement causes control to be transferred to the interpreter when the independent variable (T) reaches 1.0. In response to the interpreter's request, the analyst requests the terminal value of y with the statement:

```
Y:
```

To which, in this case, the interpreter responds:

$$Y = -0.08524$$

Recognizing that this is too low (the desired $y(t_f)$ is 1.0) and hoping for a monotonic increasing relationship between $y(t_f)$ and $y(0)$, the analyst commands the interpreter:

DYO = 2.0, GO;

This causes another solution to be run. The process of readout, alteration and run is iterated following a standard binary search algorithm; it is found that a $\dot{y}(0)$ of 2.472 yields a $y(t_f)$ of 1.0001 which is deemed satisfactory.

The SAVE control statement in the program has been storing the results (Y,T) of each iteration. Once the analyst is satisfied with the solution, he types in:

GRAPH [T,Y],CONTROL;STOP;

which produces a plot of y versus t (from the last iteration) on the I/O station; the STOP command terminates the job.

Explicit Structure Program

The preceding program was designed to assist a human analyst to solve a complicated search problem. The immediate reaction is to program the complete solution using the same algorithm (i.e. binary search). If derivative information were available however (i.e. the dependence of $y(t_f)$ on $\dot{y}(0)$, a Newton Raphson search algorithm with its associated quadratic convergence could be employed.

As illustrated in Ref. 5, the desired derivative information ($\frac{\partial y}{\partial \dot{y}(0)}$) is easily obtained by solving the auxiliary differential equation:

$$\ddot{u} = - (e^y)u$$

$$u(0) = 0, \dot{u}(0) = 1$$

$$\text{where } u(t) = \frac{\partial(y)t}{\partial \dot{y}(0)}$$

The iterative algorithm employed in the following program involves repetitively solving the differential equations in y and u adjusting the initial condition $\dot{y}^k(0)$ at each iteration k according to the equation:

$$\dot{y}^k(0) = \dot{y}^{k-1}(0) + \frac{y^{k-1}(t_f) - y^{k-2}(t_f)}{u^{k-1}(t_f)}$$

PROGRAM

```

≡ EXPLICIT STRUCTURE EXAMPLE ≡
≡ PARAMETER INFLUENCE COEFFI-
  CIENT ITERATIVE SOLUTION OF |||
  NONLINEAR TWO POINT BOUNDARY
  VALUE PROBLEMS ≡
TITLE [EXPLICIT STRUCTURE EXAMPLE ≡
  — J.C. STRAUSS]
DATA [Y0,YT,DYK,EPS,KMAX,TF/0,1,1,≡
  1E—5,10,1.0]
INITIAL
  K = 0; FLAG = .FALSE.
  INTERPRETER [Y0,YT,DYK,EPS,KMAX,≡
    Y,TF]
  YD = YT
1 K = K + 1
  PRINT 10, K
  IF (FLAG) PRINT 11
10 FORMAT (12H0 ITERATION, 31)
11 FORMAT (30HI T X U)
END
DYNAMIC
  IF (FLAG) PRINT 12,T,X,U
  IF (FLAG) CALL SAVE (X,T)
12 FORMAT (3E12.5)
  IF (T .GT. TF) GO TO 2
DERIVATIVE
  COMDEL [0.1]
  DEFINE [Y0,DYK]
  Y = INTEG [Y0,INTEG [DYK,1 + FUN]]
  FUN = EXP [Y]
  U = INTEG [0,INTEG [1.0,FUN*U]]
END
END
TERMINAL
2 DDY0 = (Y-YD)/U
  YD = Y
  DYK = DYK + DDY0
  PRINT 13, DDY0,DYK
13 FORMAT (14H0 DELTA DY0 = ,E11.5,11H
  NEW DY0 = ,E11.5)
  IF (ABS(Y-YT) .GT.EPS) .AND. (K .LT.
  KMAX)) GO TO 1
  IF (FLAG) GO TO 4
  IF (K .GE. KMAX) GO TO 3
  FLAG = .TRUE.
  GO TO 1
3 PRINT 14

```

```

14 FORMAT (54HO K .GE. KMAX — KMAX
      MAY BE INCREASED FROM INTER-
      PRETER)
      GO TO 5
4 CALL PLOT (T,X)
5 CONTINUE
END
END

```

This program is designed to be completely automatic in operation with the additional feature that parameters of the problem such as t_f , $y(t_f)$, $y(O)$ etc. can be easily varied at runtime. The interpretive command sequence to run the standard problem (using the data values supplied by the DATA control statement) is:

```

GO;
{the problem runs and produces satisfactory output}
STOP;

```

The program contains a number of organizational features worthy of more detailed discussion. In particular:

1. In an explicit structure program, the INTERPRETER control statement indicates the position in the program from which control is transferred to the interpreter. The variables specified in the argument list are made available for communication at runtime. In the case of an implicit structure program, the whole symbol table of the source program is passed to the interpreter.
2. The translator automatically closes the control loop around the dynamic region. Hence it is absolutely necessary that the programmer provide explicit termination logic via a procedural statement(s).
3. All output is performed with procedural (Fortran) statements although BHSL control statements could have been used.
4. The iteration logic is programmed such that the solution trajectory is printed out on an extra iteration following satisfaction of the convergence test.

5. Should the maximum number of iterations (KMAX) be exceeded, an error message is printed and control is returned to the interpreter where the analyst can take a variety of actions. Barring just terminating the problem, the simplest action would be to increase KMAX and continue. He could however change the initial guess on $\dot{y}(\emptyset)$ (DYK) and restart the iterative process.

6. The DEFINE statement in the derivative section names those variables whose values are defined external to the derivative section. There are two reasons for this: 1) The derivative sections are processed first since they determine the storage allocation for the object program. The define statement must specify those variables which are considered to be defined for sort purposes. 2) The translator does not scan the procedural statements and is thus not aware of any variables whose values are determined by procedural coding.

REFERENCES

1. R. D. Brennan and R. N. Linebarger, "A Survey of Digital Simulation: Digital-Analog Simulator Programs," *Simulation*, vol. 3, no. 6 (Dec. 1964), pp. 23-36.
2. J. J. Clancy and M. F. Fineberg, "Digital Simulation Languages, A Critique and a Guide," AFIPS Volume 27, 1965 Fall Joint Computer Conference, Spartan Books, Washington, D. C., 1966.
3. "Continuous System Simulation Language," Report of the SCI Simulation Software Committee; J. C. Strauss, Ed., presented at SCI 2nd Annual Simulation Software Meeting, Minneapolis, Minn., June 1966. (to be published in *Simulation*)
4. M. L. Cramer and J. C. Strauss, "A Hybrid-Oriented Inter-Active Language," 1966 National ACM Conference, Los Angeles, California, August 1966.
5. G. H. Burgin, "MIDAS III . . . A Compiler Version of MIDAS," *Simulation*, vol. 6, no. 3 (March 1966).

A PROCESSOR-BUILDING SYSTEM FOR EXPERIMENTAL PROGRAMMING LANGUAGES

Terrence W. Pratt

Michigan State University, East Lansing, Michigan

and

Robert K. Lindsay

University of Michigan, Ann Arbor, Michigan

INTRODUCTION

Translator-building systems which allow the rapid construction of translators for programming languages have been in existence for a number of years, beginning with pioneering efforts by Irons¹ and Brooker and Morris,² and more recently in systems developed by Reynolds,³ McClure,⁴ and Feldman,⁵ among others. With such systems it is possible to build a translator for a language relatively easily, although the translator may not be a particularly efficient one. In this paper, an extension of the notion of a translator-building system to the notion of a processor-building system is considered, and an operating example of such a processor-building system is described.

A system which accepts programs written in a particular programming language as input and then executes those programs may be termed a *processor* for that language. The processing of programs is commonly divided into two more-or-less distinct phases—*translation* and *execution*. In the translation phase programs written in the input or “source” language arrive in the form of character strings from

some input device. They are translated (by a *translator*) into an intermediate form, which may be machine code or some “high-level” form. This intermediate program form serves as the input for the execution phase, in which this form of the program is interpreted (by an *interpreter*) and the operations on the data specified by the program are executed.

A general form for such a processor is outlined in somewhat greater detail in Fig. 1. As input we have programs and data described in the source language. From this input the translator produces an intermediate form of the program and an internal form of the data. We then have an interpreter together with a set of subroutines called *basic processes* (which manipulate data, and programs if considered as data). The interpreter accepts programs in the intermediate form as input and calls the appropriate sequence of basic processes to execute the instructions of the program.

To define a processor for a programming language L, then, one might define: (1) A translator for L which accepts L programs and data as input and produces an intermediate form of these programs and data as output; (2) a set of basic proc-

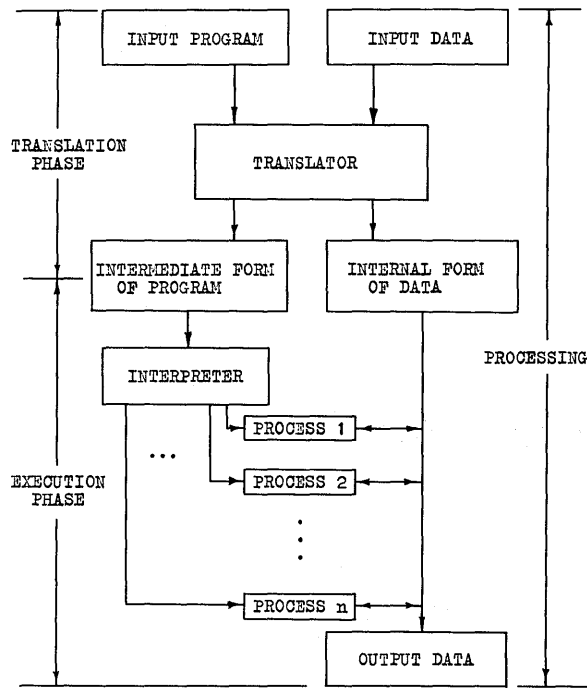


Figure 1. A processor for a programming language.

esses for performing the operations which can be specified in L programs; and (3) an interpreter which accepts L programs in the intermediate form as input and interprets them as indicating the basic processes to be called, the sequence of these calls, and the input parameters for each call.

In translator-building systems such as those mentioned previously, the user may conveniently describe the translation phase of the processing of his source language. Because the user does not have a similar ability to build an interpreter for the resulting output, however, the output from such a system must be in a form acceptable to an existing interpreter (such as the machine itself), or it must be acceptable as input to another existing processing system (such as an assembler). Restrictions of this sort on the output form of a translation may significantly increase the difficulty of building translators for new languages, as there may be no straightforward way to translate a new language into one of these relatively fixed forms. To achieve a rapid implementation of a new language it may be easier in many cases to use a relatively simple translation from input language into high-level intermediate form, and then interpret the resulting intermediate form with a correspondingly "high-level" interpreter, in preference to performing a complex

translation to one of these fixed output forms before beginning execution. Thus a processor-building system with which interpreters as well as translators may be constructed is a desirable extension of the notion of a translator-building system.

We are now in a position to describe a processor-building system which does allow the user to build an entire processor for a programming language rather than just a translator. The system, called AMOS, is programmed and running on a Control Data 3600 computer. The input to the system is in two parts—a *translation phase* definition and an *execution phase* definition. After the system processes these two sections of input, it is prepared to process programs in the specified source language, first translating the programs according to the specifications of the translation phase definition and then executing the resulting output according to the specifications of the execution phase definition. The translation phase definition, which is discussed in the next section of this paper, is similar in general approach to other translator-building systems, although the details differ considerably. The execution phase definition is described in the third section of the paper.

TRANSLATION PHASE DEFINITION

As the notion of a translator-building system which produces syntax-directed translators is well known, it will suffice to merely outline the general approach here before considering the particulars of the AMOS system. The general pattern of such systems is the following. The syntax of the source language to be implemented is described with a grammar. With each grammar rule a semantic routine or "interpretation routine" is associated, which describes the output or intermediate processing which is to occur should the syntactic construct described by the grammar rule occur in a source language program. The grammar rules and interpretation routines are the initial input to the translator-building system. The system stores this information in a convenient form so that it can be used later to allow the system to translate programs in the user-defined source language. The basic concepts of translator-building systems are described in a recent paper by Cheatham.⁶ Rather than proceeding further in describing these basic ideas, we shall instead concentrate on the distinctive features of the AMOS system. The basic structure of the translation definition in AMOS is

similar to that described above. The four headings under which features of this part of the AMOS system will be discussed are: (1) the initial loader, (2) the grammar form and the parsing algorithm, (3) the form of the output from translation, and (4) the language for writing interpretation routines.

The Initial Loader

All input to AMOS—grammar rules, interpretation routines, processor definitions, and source language programs—is processed first by the initial loader, described in detail in Doig⁷ and Pratt.⁸ The loader has two main responsibilities. It must translate the character codes used for input on the particular machine configuration being used into a standard set of AMOS character codes. Thus to the rest of AMOS, the character code for “\$,” for example, is fixed, regardless of the original input code used. By making simple changes in the loader tables, the system can be adapted to a different set of character codes in a matter of minutes. The second responsibility of the loader is that of editing the input string while it is being translated into the AMOS character codes. For the grammar rules, interpretation routines, and the processor definition, this editing amounts merely to the deletion of spaces from the input string. For the source language programs supplied by the user, however, the editing may be more significant. The user has control, through use of a FORMAT statement, of the editing to be performed on his program. The possible options include: (1) deletion of all occurrences of a particular character; (2) reduction of strings of occurrences of a particular character to a single occurrence; (3) deletion of comments enclosed by a designated “comment delimiter”; (4) specification of input records of fixed length; and (5) specification of fields within such fixed length records.

Options (4) and (5) allow fixed field card formats to be used in the source language. The loader compacts such fixed field formats by: (1) deleting all characters not within the specified fields, (2) adding a special “end-of-field” delimiter at the end of the characters in each field, and (3) adding an “end-of-record” delimiter at the end of each record. Thus field and record boundaries are represented by explicit characters in the edited string. This allows grammar rules to consider these boundary markers as syntactic elements of the program.

The user may change editing formats within his

source language program by insertion of a new FORMAT statement, thus allowing the mixing of free and fixed formats within the source language. In addition to making AMOS independent of the input character codes being used, the main virtue of the loader to the user is the simplification that it allows in the syntactic descriptions of the source language without restricting the actual input format used.

The Grammar Form and Parsing Algorithm

Grammars which are used to describe the syntax of source languages in AMOS are written in a recognition-oriented form called the *tactic grammar* form. A tactic grammar is composed of a finite set of grammar rules. Each grammar rule has four fields, written A/B/C/D/, where A is the *context* field, B is the *left side* field, C is the *right side* field, and D is the *interpretation* field. The context field contains a “context name” (an arbitrary character string) or it is blank, indicating that the context is the same as that of the immediately preceding rule in the list. The left and right sides contain: (1) a literal character string which may occur in the source language, (2) a context name enclosed in quotes, indicating the use of this name as a “syntactic type,” or (3) the special character Φ , denoting an arbitrary character. The interpretation field contains the interpretation routine for the grammar rule. This routine is written in the I-language described in the last part of this section and may contain in particular the grammar control statements *CONTEXT* and *ASCEND*.

In order to understand how a set of tactic grammar rules defines the syntax of a language, it is necessary to understand the parsing (recognition) algorithm used as well as the grammar form itself. An example of a simple BNF grammar for an arithmetic assignment statement and an equivalent tactic grammar is given in Table 1. The parsing algorithm used by AMOS is diagrammed in Fig. 2.

Basic to the notion of parsing with a tactic grammar is the notion of the “current context.” At all times during the course of parsing there is a current context which delimits the set of rules which are “potentially applicable” at any given moment. A tactic grammar rule A/B/C/*CONTEXT*D./ may be interpreted as meaning: If A is the current context, then a B can be followed by a C. If this construction is found and the rule is “applied” (that is, if a B is followed by a C in the input string),

Table 1. Equivalent BNF and Tactic Grammars for an Arithmetic Assignment Statement

BNF Grammar	Tactic Grammar
$\langle \text{stmt} \rangle ::= \langle \text{id} \rangle = \langle \text{expr} \rangle$	STMT / \emptyset / 'ID' / /
$\langle \text{expr} \rangle ::= \langle \text{term} \rangle \langle \text{expr} \rangle + \langle \text{term} \rangle$	/ 'ID' / = / /
$\langle \text{term} \rangle ::= \langle \text{fact} \rangle \langle \text{term} \rangle * \langle \text{fact} \rangle$	/ = / 'EXPR' /*ASCEND*/
$\langle \text{fact} \rangle ::= \langle \text{id} \rangle (\langle \text{expr} \rangle)$	EXPR / \emptyset / 'TERM' / /
$\langle \text{id} \rangle ::= \langle \text{letter} \rangle \langle \text{id} \rangle \langle \text{letter} \rangle$	/ 'TERM' / + / /
	/ + / 'TERM' / /
	/ 'TERM' / \emptyset /*ASCEND*/
	TERM / \emptyset / 'FACT' / /
	/ 'FACT' / * / /
	/ * / 'FACT' / /
	/ 'FACT' / \emptyset /*ASCEND*/
	FACT / \emptyset / 'ID' /*ASCEND*/
	/ \emptyset / (/ /
	/ (/ 'EXPR' / /
	/ 'EXPR' /) /*ASCEND*/
	ID / \emptyset / 'LET' / /
	/ 'LET' / 'LET' / /
	/ 'LET' / \emptyset /*ASCEND*/

then the current context becomes D (or stays the same if no *CONTEXT* statement occurs in the interpretation field). Thus when A is the current context and a B has just been recognized in the input string, the set of "potentially applicable" rules is just the set of rules having A in their context field and B in their left side field. The next rule to be applied must be chosen from this set.

During parsing the current context may be changed by applying a rule which has a CONTEXT or ASCEND statement in its interpretation field. The CONTEXT statement is used to change the current context directly to a specified new context. The ASCEND statement on the other hand is used when a string of a particular syntactic type has been recognized in the input. Execution of this statement returns the current context to whatever it was before "descending" to recognize the syntactic type. The initial "current context" for a grammar is specified in an initialization routine preceding the first grammar rule. This routine is executed immediately before parsing begins.

The parsing method specified in detail in Fig. 2 involves a single left to right scan of the input string. Two pushdown stacks are used to allow a simple type of backtracking. No "parse tree" is constructed, and there is no rewriting of the input string. The stacks are used to save the position of the input string pointer and the location of the current rule

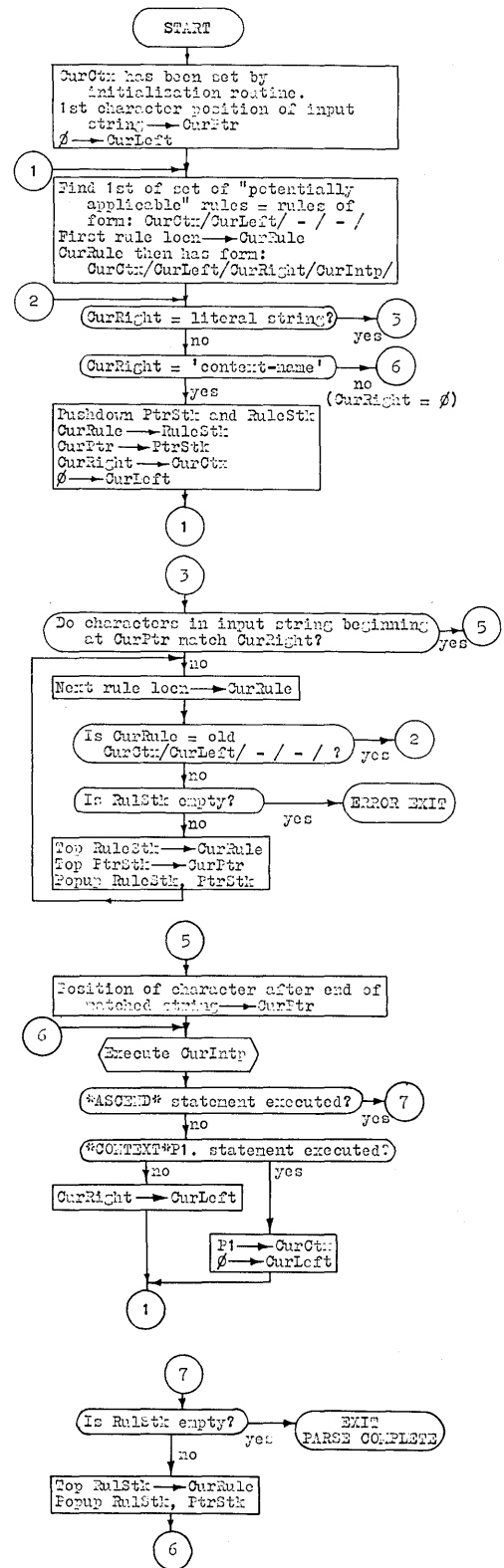


Figure 2. Parsing algorithm for tactic grammars. NOTE: Any change in CurRule redefines CurCtx, CurLeft, CurRight, and CurIntp to have the values of the new rule named in CurRule.

when descending a level in parsing to attempt to recognize a string of a certain syntactic type. While at the lower level the input string pointer may be advanced, but if eventually the type is not recognized, then the pointer saved in the pointer stack replaces the current input string pointer upon return to the higher level, thus allowing a simple sort of backtracking. The rules stack is used only to save the location of the rule with which a match is being attempted when such descents are made; it is not used to hold the locations of all rules which have been applied in the parsing. The parsing scheme is straightforward, and quite efficient since there is: (1) only simple backtracking, (2) no storage of information about the course of the parse except what is absolutely necessary to allow the parse to continue, and (3) only a small set of "potentially applicable" rules to be tried at any point. The interpretation routine for each rule used in the parse is executed at the time the rule is applied. Thus translation proceeds in parallel with parsing.

In a translator-building system, the grammar form and the parsing algorithm which are used are significant because they delimit the set of input languages which it is possible to describe to the system and which may be parsed correctly by it. Thus it would be more to the point perhaps to describe the restrictions on the class of possible input languages imposed by the grammar form and parsing algorithm used in AMOS. Unfortunately this is difficult to do. Beyond the fact that the class of languages describable by tactic grammars includes the finite-state languages, not much is known about this class except that some rather complex languages have been described with tactic grammars. One of the interesting aspects of attempting to characterize the class of languages describable by tactic grammars in this form is the fact that the specification of source language syntax contained in the grammar rules is not completely independent of the specification of semantics in the interpretation routines for the rules. The grammar control statements `CONTEXT` and `ASCEND` appear in the interpretation routines but are actually a part of the syntactic description of the source language. The ability to branch in an interpretation routine and execute different `CONTEXT` or `ASCEND` statements, depending on semantic information available to the interpretation routines, means that one may make the allowable syntax of the source language dependent on semantic information. One interesting aspect of this fact in

particular is that statements at the first portion of a source language program may be used to dynamically change the allowable syntax of later portions of the same program.

In summary, an interesting class of languages is describable with tactic grammars, although no precise characterization of this class has yet been made; and in addition interaction between syntax and semantics in the translation description is allowed in what seems to be a novel manner.

Outputs from Translation

The grammar form and parsing algorithm delimit the class of *input* languages acceptable by AMOS. The *output* from translation is similarly circumscribed by the type of data structures which can be produced in AMOS to represent intermediate forms of programs and data.

The elementary unit of output from the translation phase is termed a *record*. A record is composed of a consecutive sequence of bits in memory, logically divided into *fields*. Each field in a record may be of arbitrary length, and may be either a simple field, or an array field. If an array field, then all elements of the array are the same size. Records may be generated in sequential locations in memory, or they may be linked together in an arbitrary fashion by storing the address of one record in a field (termed a link field) of another record. As records may have any number of link fields, linked structures may be built up which correspond to general finite directed graphs. Besides link fields, records may also have any number of other fields containing data.

Records are defined by first defining the *pattern* of the fields of the particular type of record. This pattern is named, and the form of each field is defined by specifying the length of the field in bits, the name of the field if any, whether or not it is an array, and if so its dimensions. After having defined a pattern (and any number of different patterns may be in existence concurrently), records may be created which have that pattern. When a record is created, the number of bits associated with records of the specified pattern type are reserved in memory. One may then refer to fields within the record simply by giving the record name and the name of the desired field. The statements for defining patterns, fields, and records are described in the next section. One may also specify that records of a certain type are

“sequential”. This implies that whenever a record of that type is created, the block of space reserved for it is to come immediately after the block occupied by the last record defined of that type. Thus if the pattern “machine-code-word” is specified as “sequential”, then all records of type “machine-code-word” will be stored in memory in consecutive locations in the order created. Both patterns and records may be created dynamically during the course of translation (or execution). The type of data which may be stored in fields of records includes numbers (integers or reals), addresses, or character strings.

The output from translation, therefore, is general enough to allow the user considerable flexibility in choosing the sort of translation which he wishes to make. In the case of input *data* to be translated, this makes possible the construction of data structures of considerable complexity during translation. In the case of input *program*, this general output, in conjunction with the ability to describe an interpreter for programs in arbitrary form described below, under “Execution Phase Definition,” allows the user considerable flexibility in choosing the simplest processing method for his language.

The I-Language for Writing Interpretation Routines

The I-language is a simple programming language used in writing interpretation routines for tactic grammar rules as well as writing interpreters and basic processes in the execution phase definition described in the next section. Statements in I-language routines all have the standard form: *operation-name*parameter-list. The operations are called primitive processes or simply *primitives*. Table 2 contains a list of the primitives currently being used in the I-language with a brief description of the function of each. An I-language routine is just a sequence of I-language statements written free-field and ending with a “\$.” Routines may be named and called from other routines, statements within routines may be labeled, and branching is allowed.

Table 3 gives an example of a simple translator definition. The translator accepts an arithmetic assignment statement as input according to the syntax described by the grammar of Table 1. The translation is into a simple suffix form composed of a linked list containing in sequence the operands and operators of the suffix form with a flag bit in each

Table 2. Brief Descriptions of I-Language Statements Currently in Use

<i>I-Language Statement</i>	<i>Function</i>
*CONTEXT*P1.	Change current context to P1.
ASCEND.	Syntactic type recognized, ascend a level in parsing.
*TRANSFER*P1,P2.	Copy the contents of P1 into P2.
*PATTERN*P1,P2.	Begin definition of a pattern named P1.
*FIELD*P1,LENGTH(n), ARRAY(P2).	Define a field in the current pattern under definition.
*DEFINE*P1,P2,P3.	Define a record of pattern type P1, store its address in P2, and (optionally) name it P3.
*TEST*P1,P2,P3,P4.	If the contents of P1 is positive, zero, or negative, branch to P2, P3, or P4 respectively.
*GO TO*P1.	} Sequence control in I-language routines.
EXIT.	
*CALL*P1.	Execute routine P1.
*ADD*P1,P2,P3.	} Arithmetic operations: Compute P1 op P2 and store the result in P3.
*MULT*P1,P2,P3.	
*SUB*P1,P2,P3.	
*DIV*P1,P2,P3.	
*DATALIST*P1, . . . Pn.	Define P1 . . . Pn as house-keeping lists for temporary storage.
*POPOP*P1, . . . Pn.	} Popup or pushdown the house-keeping lists designated by P1 . . . Pn.
*PUSHDOWN*P1, . . . P1	
*PRINT*P1,P2.	Print the contents of the field P2 according to the format P1.
FIRSTCHAR.	Clear the buffer DATA and begin building up a string there by entering into the buffer the character in the input string at the current pointer position.
NEXTCHAR.	Add the character in the input string at the current pointer position to the string being formed in DATA.
*CONVERT*P1.	Convert the string in the buffer DATA into internal form for data of type P1 (integer, real, or address).

cell to distinguish operators from operands. The list cell pattern is simply:

F	OP	LINK
---	----	------

where OP and LINK contain addresses and F=1 implies OP contains the address of an operand, F=0 implies OP is the address of an operator rou-

Table 3. Tactic Grammar of Table 1 with Interpretation Routines for Producing Suffix Form Lists

```

(initialization routine)
*PATTERN*LISTWD. *FIELD*F,LENGTH(1). *FIELD*OP,LENGTH(15).
*FIELD*LINK,LENGTH(15). *PATTERN*DATAWD. *FIELD*DATAH,LENGTH(49).
*CONTEXT*STMT. *DATALIST*TEMP,CIA.  § /
(grammar rules)
STMT/  § / 'ID' / *DEFINE*LISTWD,TEMP(1). *TRANSFER*TEMP(1),
        CIA(1). *CALL*SUB2.  §/
/ 'ID' / = / § /
/ = / 'EXPR' / *CALL*SUB1. *TRANSFER*ASSIGN,(TEMP(1),OP).
        *ASCEND*.  §/
EXPR/  § / 'TERM' / § /
/ 'TERM' / + / § /
/ + / 'TERM' / *CALL*SUB1. *TRANSFER*ADDFUNC,(TEMP(1),OP).  §/
/ 'TERM' / § / *ASCEND*.  §/
TERM/  § / 'FACT' / § /
/ 'FACT' / * / § /
/ * / 'FACT' / *CALL*SUB1. *TRANSFER*MULTPROC,(TEMP(1),OP).  §/
/ 'FACT' / § / *ASCEND*.  §/
FACT/  § / 'ID' / *CALL*SUB1. *CALL*SUB2. *ASCEND*.  §/
/ § / ( / § /
/ ( / 'EXPR' / § /
/ 'EXPR' / ) / *ASCEND*.  §/
ID /  § / 'LET' / *FIRSTCHAR*.  §/
/ 'LET' / 'LET' / *NEXTCHAR*.  §/
/ 'LET' / § / *CONVERT*NAME. *ASCEND*.  §/
DUMMY/ / / SUB1: *DEFINE*LISTWD,TEMP(2). *TRANSFER*TEMP(2),
        (TEMP(1),LINK). *POPUP*TEMP.  §/
/ / / SUB2: *TRANSFER*1,(TEMP(1),F). *DEFINE*DATAWD,TEMP(2),
        DATA. *TRANSFER*TEMP(2),(TEMP(1),OP).  §/

```

NOTES

The blank rules in context DUMMY are never used; this is merely a device to allow definition of the subroutines SUB1 and SUB2.

The names ASSIGN, ADDPROC, and MULTPROC are names of basic processes defined in Table 4.

References to housekeeping lists of the form "h-k-list-name(n)" are references to the contents of the *n*th cell on the designated list (e.g., TEMP(2) refers to the contents of the second cell on the housekeeping list TEMP).

A reference of the form "(h-k-list-names(n),field-name)" is an indirect reference to the contents of the designated field of the record whose address is stored in the designated housekeeping list location (e.g., "(TEMP(1),OP)" refers to the contents of the OP field of the record whose address is stored in TEMP(1)).

tine. LINK contains the address of the next cell on the list. In the next section an interpreter for this output will be constructed.

The basic design philosophy of the I-language has included two major goals: (1) The I-language should be as easy to learn and use as possible, and (2) every effort should be made to make it easy to

add new primitive operations to the language. The combination of these two goals has led to the implementation of a basic set of primitives with provision for the addition of new primitives when necessary. For experimental languages it does not seem desirable or even possible to develop a rigid I-language for use in all applications. Thus a simple basic language has been implemented with provision made for easy modification later to fit particular types of applications. The I-language as specified in Table 2 has proven sufficient for describing a translation-execution processor for a hierarchical-graph-structure manipulating language of some complexity. Thus, while simple, the I-language is still quite powerful, so that additions to fit particular applications, if any, will in general be relatively minor.

This completes the description of the translation phase definition. In summary, to construct a translator with AMOS, the user writes a grammar for his source language in the form of a tactic grammar and describes the translation of his source language into the desired output form with I-language routines written in the interpretation field of each grammar rule. Adding an initialization routine (again written in the I-language) completes the description of the translation phase. From this information AMOS constructs a translator for source language programs. The output from this translation might be saved for later input to some standard processor, but in general it would be executed by an interpreter defined in the execution phase definition described in the next section.

EXECUTION PHASE DEFINITION

The output from the translation phase of a processor is an intermediate form of the source language program and data which was originally input. In the execution phase of processing, this intermediate form of the program is interpreted by some interpretive routine and the specified operations executed. Thus the output from the execution phase of processing is the input data transformed according to the specifications of the original program. One way of structuring the execution phase (although not the only one) is to assume that there are a set of *processes* defined which perform the basic operations which can be specified in the source language. Depending on the type of data and the sort of

operations possible, these basic processes might correspond to arithmetic operations, list operations, logical operations, tests, searches, etc. The intermediate form of the program is then an encoded specification indicating which of these basic processes are to be executed, in what order, and with what parameters. The function of the interpreter is to decode the intermediate form of the source language program and to actually call the designated sequence of basic processes after setting up their parameters appropriately. Examples of this sort of organization are seen in the hardware of many computers, where the machine code may be considered as the intermediate form of the program, the "wired-in" basic processes are the circuits which execute the basic machine instructions, and the interpreter, which is also "wired-in," decodes each machine instruction and calls the appropriate basic process depending on the operation code of the instruction. This organization has been used when working with high-level intermediate languages as well (e.g., this is the basic organization of the IPL-V system).

The execution phase of language processing in AMOS is organized in this manner. Basic processes are data manipulation routines with parameters, while an interpreter is a routine which accepts data structures representing programs as input, and calls various of the basic processes in sequence to process the data. Thus the definition of the execution phase involves the construction of routines for the interpreter and basic processes. As the I-language is a programming language which is already available in the AMOS system, it is used in writing the routines for the interpreter and basic processes. An example is given in Table 4 of a simple execution phase definition for executing the suffix form output from the translator of Table 3. Since the form of routines in the execution phase definition is the same as the form of the interpretation routines associated with grammar rules, the definition of such routines given in section II also applies to routines in the execution phase definition.

In summary, to construct a processor for a programming language with AMOS, the user prepares input in two parts: (1) The tactic grammar and interpretation routines defining the translation phase and (2) the routines defining the interpreter and basic processes of the execution phase. After AMOS has processed this input, it is prepared to process programs—translate and then execute them—in the

Table 4. Interpreter and Basic Processes for Execution of Suffix Form Lists

```
(interpreter)
RINTERP: *DATALIST*CIA,STACK,TEMP.
LOOP: *TEST*(CIA(1),F),OPND,OPTR,OPND.
OPND: *PUSHDOWN*STACK. *TRANSFER*(CIA(1),OP,STACK(1)).
      *GO TO*NEXT.
OPTR: *CALL*(CIA(1),OP).
NEXT: *TEST*(CIA(1),LINK),NXT,END,NXT.
NXT: *TRANSFER*(CIA(1),LINK),CIA(1). *GO TO*LOOP.
END: *PRINT*INTEGER,(STACK(1),DATUM). §

(basic processes)
ADDPROC: *DEFINE*DATAID,TEMP(1). *ADD*(STACK(1),DATUM),
        (STACK(2),DATUM),(TEMP(1),DATUM). *POPOP*STACK.
        *TRANSFER*TEMP(1),STACK(1). §
MULTPROC: *DEFINE*DATAID,TEMP(1). *MULT*(STACK(1),DATUM),
        (STACK(2),DATUM),(TEMP(1),DATUM). *POPOP*STACK.
        *TRANSFER*TEMP(1),STACK(1). §
ASSIGN: *TRANSFER*(STACK(1),DATUM),(STACK(2),DATUM). *POPOP*STACK. §
```

defined source language. Source language programs would then comprise a third section of input.

CONCLUSION

The construction of AMOS has been motivated by the desire to find a method to implement a new language as quickly as possible on a computer which would also allow changes to be made easily in that implementation or in the source language. One inherent difficulty in systems which allow the user to specify only the translation phase of processing—a difficulty that AMOS attempts to circumvent—lies in the fact that the output from translation must then be in a form acceptable to an existing processing system. It is a basic thesis here that there exists a significant class of languages for which no simple translation into such a form exists, but for which one might invent an output form which could be easily interpreted and executed. For such languages it will be simpler to define an entire processor than to define a translation into a form acceptable to an existing processor.

In the description of the system presented here, much has necessarily been omitted. The translator-building portion of the system may be used independently if desired, and the output from translation executed by an existing processor. In general all parts of the system are organized in a flexible manner so that changes and additions are easily introduced. As the system is intended as an experimental tool it contains few frills for the user. In addition no attempt has been made to provide sophisticated so-

lutions to such problems as storage allocation and input/output handling.

Forms of the AMOS system have been coded and running on a Control Data 3600 computer since about October 1965, although the execution definition section has only been added recently. Experience with the system so far has been limited. The major language implemented to date with AMOS is HINT—a language designed by Richard Hart to be used for manipulating complex hierarchical-graph structures. In implementing this language, AMOS was used to build a translator to translate HINT programs into IPL-V list structures, which were then interpreted and executed by the IPL-V interpreter. Experience in teaching the use of AMOS to students has indicated that it is easy to learn to use, requiring only a few hours of instruction before one can begin to write translators. Results from further experience with the system will be reported at a later date.

ACKNOWLEDGMENTS

An early version of a translator-building system known also as AMOS was constructed by Kenneth M. Shavor and the authors. A number of the ideas developed in that system have been incorporated in

the AMOS system described here.

REFERENCES

1. E. T. Irons, "A Syntax Directed Compiler for ALGOL 60," *Comm. ACM*, vol. 4, no. 1 (1961).
2. S. Rosen, "A Compiler-Building System Developed by Brooker and Morris," *ibid.*, vol. 7, no. 7 (1964).
3. J. C. Reynolds, "An Introduction to the COGENT Programming System," *Proc. ACM 20th Nat'l Conf.*, Aug. 1965.
4. R. M. McClure, "TMG—A Syntax Directed Compiler," *ibid.*
5. J. A. Feldman, "A Formal Semantics for Computer Languages and Its Application in a Compiler-Compiler," *Comm. ACM*, vol. 9, no. 1 (1966).
6. T. E. Cheatham and K. Sattley, "Syntax-Directed Compiling," *Proc. AFIPS Spring Jt. Comp. Conf.*, 1964.
7. L. Doig, R. Elliot and T. Pratt, "Load Program for the MO Compiler," *Info. Proc. Report No. 6*, Univ. of Tex. Comp. Center (Feb. 1964).
8. T. Pratt, "Syntax-Directed Translation for Experimental Programming Languages," Univ. of Tex. Comp. Center, TNN-41 (May 1965).

THE INTRODUCTION OF DEFINITIONAL FACILITIES INTO HIGHER LEVEL PROGRAMMING LANGUAGES

T. E. Cheatham, Jr.

*Massachusetts Computer Associates, Inc.
Wakefield, Massachusetts*

INTRODUCTION

The purpose of this paper is to present a scheme for employing definitional or "macro" features in a higher level programming language. The emphasis will not be on defining the syntactic augments and precise interpretation of such features in any particular programming language and/or operating environment but, rather, on developing the compiler mechanisms for handling the definition and call of such macros and then indicating the kinds of extensions one might propose to current programming languages in order to usefully employ these kinds of facilities.

Macro facilities have been incorporated for some time in most machine language assembly systems. More recently, macro systems have been developed for string and text handling both as independent systems^{1,2} as well as within the framework of some other systems. However, there have been relatively few attempts to incorporate the ability to extend a higher level programming language via macro type facilities. Some exceptions to this are the DEFINE facility included in the various JOVIAL languages³ which allowed the substitution of an arbitrary character string upon each occurrence (during lexical analysis) of an identifier previously DEFINED. The PL/I language⁴ has a quite elaborate facility for text-editing type macros which are employed prior

to lexical and syntactic analysis of the source text (and which thus require special lexical and syntactic analysis machinery to be available as a pre-processor); further, the PL/I facility for definition of "arbitrary linear mapping function" is a primitive macro facility which is employed after syntactic analysis. The proposal by Galler and Perlis⁵ suggests an interesting extension to ALGOL to allow for user-controlled syntactic and semantic augments to ALGOL.

One of the reasons for the lack of macro facilities in higher level programming languages may be that we can identify at least four distinct kinds of macro facilities which might be introduced, each with quite definite advantages and disadvantages. Thus, we submit that to speak of "adding macro facilities" is merely confusing; one must indicate with some precision just where, when, and how he proposes to add such facilities. Very briefly, we identify three times during the compilation process where macros might be added as follows:

Preceding Lexical Analysis—for editing text and incorporating filed or otherwise prepared text into a program prior to any lexical or syntactic analysis of the program text.

During Syntactic Analysis—allowing the introduction of new syntactic structure and

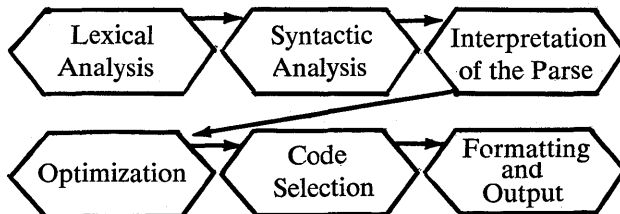
the corresponding semantic structure into the language.

Following Syntactic Analysis—particularly useful for introducing “open coding” for various procedures and functions, such as mapping functions for array elements.

Historically, the schemes and mechanisms we discuss in this paper have arisen out of our experience in dealing with a variety of compiler-compilers (or, more properly, compiler-building systems) and using them to construct translators for a variety of standard and special-purpose programming languages, including ALGOL and PL/I. The compiler-building system on which most of the ideas developed herein are based is the TRANGEN system and its compiler-building language TRANDIR.^{6,7}

As indicated above, we are not going to frame our discussion around any particular programming language; however, if a focus is desired, our thinking is more often than not with respect to languages like ALGOL or PL/I. When we need to reference the programming language being compiled, we shall refer to it simply as language L_P . The ideas will be developed with reference to a proposed *compiler model* and *compiling system* which are more-or-less easily particularized to most of the current programming languages. By a compiler model we mean a division of the compiling task into a collection of conceptually distinct parts or phases; we are not suggesting that every compiler must be constructed with these phases, nor do we regard the phases as corresponding to “memory loads” or “tape movements.” Rather, they are conceptual divisions of the compiling task into pieces we choose to distinguish and with respect to which we will explain the various definitional facilities with which this paper is concerned.

The parts of the compiler model can be represented by the following diagram:



A brief description of the function of each of the parts follows:

Lexical Analysis—the process of interfacing with the source of program text and identifying, isolating, and disposing of the

terminal symbols and *token symbols* occurring in the program text, and outputting a sequence of *descriptors* of these terminal and token symbols. By terminal symbols we mean the basic characters and strings of characters comprising the alphabet of the language, e.g., “+”, “;”, “begin”, “procedure”, and the like; by token symbols we have in mind the strings of characters comprising individual members of such (terminal) classes as identifiers and literals (e.g., “ALPHA”, “1.5604E-3”, “true”, and the like). By “converting” and “disposing of” the token symbols we mean whatever process is required to provide us with an entry in a symbol table or a literal table and produce the resultant descriptor pointing to that entry.

Syntactic Analysis—the process of identifying the syntactic units (with respect to some particular syntax) occurring in the stream of descriptors produced by the lexical analysis process, resulting in a parse of the program text.

Interpretation of the Parse—the process of generating that representation of the computation which will provide the basis for whatever optimization and subsequent generation of machine (or other) coding is to follow. In this paper we will restrict ourselves to a “pseudo-code” or “computation tree” representation of the computation; prefix or suffix representation, direct machine code representation, and other variations are possible with the mechanisms described, but we shall not here consider them further.

Optimization—the process of preparing the pseudo-code representation and gathering a variety of useful information prior to the actual generation of machine coding. Such things as elimination of common sub-expressions, flow analysis, development of statistics to guide register allocation and so on are included here.

Code Selection—the process of inspecting the pseudo-code representation of the computation and, using the information developed during the optimization phase, generating the sequence of machine coding for carrying out the computation.

Formatting and Output—the process of

preparing the final machine code representation (e.g., relocatable binary) and other information (e.g., some form of symbol table) from the machine code as produced by the code selection process in the formats required for punching, running, filing, or whatever disposition is to be made of the resultant program.

The *compiler* for L_P will be represented by means of "translation programs" in two programming languages which we will describe below: the base language L_B and the descriptive language L_D . We shall associate various portions of the translation programs with the various parts of the compiler model; the actual compilation will be done by the compiling system performing the set of actions or statements given in L_B and L_D by whatever means it may choose to do them (interpretively, from a machine code version resulting from compiling a program in L_B , or whatever); the only assumption we make is that any L_D program (i.e., the part of the compiler represented in L_D) is translated into an equivalent L_B program (by one of the facilities in the compiling system) prior to the translation of the L_B program into a form which will "run" the compiling system.

The base language L_B is the language in which we describe all of the manipulations which are performed on the data representing the source text to produce, in the end, the data representing the machine (or other) code result. The data being manipulated consists of a collection of *tables* in which are recorded the relevant properties or attributes of the items or symbols being manipulated (identifiers, literals, operations, and the like) and a list of *symbol descriptors*. The list of symbol descriptors represents the program text being compiled in any of its intermediate forms: string of source characters, list of lexical units, syntax tree, sequence of pseudo-instructions, sequence of machine instructions, and so on. Thus during syntactic analysis, an identifier will be represented by a symbol descriptor which points to the symbol table entry for the identifier, plus an indication of its "current syntactic type" (primary, term, expression, etc.); a computation may be represented by a symbol descriptor which points to a sequence of pseudo-instructions comprising the computation, again plus an indication of syntactic type. We may think of a symbol descriptor as a quadruple of integers: a *table* code and the *line* number within the table which contains the properties of

the symbol; a *type* field containing current syntactic type; and a single bit indicator which we shall term *larg* indicating whether or not the descriptor is the last argument of some pseudo instruction.

Language L_B can be thought as containing, as a sub-language, a more-or-less standard algebraic language including assignment statements, relationals, if-then-else statements, control transfer statements, statement labels, a block structure similar to ALGOL or PL/1, and the like. The numbers which it manipulates are integers which are fields of tables, fields of descriptors, literals, working variables, and so on. L_B has, in addition to the algebraic sub-language, a language for performing pattern matching and replacement over sequences of symbol descriptors. Much of the syntactic analysis and the code selection portions of a compiler are written using the pattern-replacement part of L_B ; however, there is a complete algebraic language available when it is desired—for handling declarations, determining fixed point scaling, error recovery, preparing relocatable binary output, and so on.

The descriptive language L_D is a declarative language in which the syntax of L_P and a certain portion of the "semantics" of L_P are given. We might think of L_D as a syntax-describing language such as BNF to which we have added facilities for representing various manipulations corresponding to the different syntactic constructions.

The *compiling system* is that collection of programs and data, under an executive system, which can carry out all the language processing, language executing, and data handling sketched above. Specifically, there will be provisions for:

1. translation of language L_D into language L_B , and subsequent editing, modification, and so on of a language L_B program;
2. translation of language L_B programs into appropriate table layouts and code (machine code, interpretive code, some mix of machine and interpretive, etc.);
3. a means for executing the code resulting from (2) (e.g., an interpreter);
4. a lexical analyzer which performs as suggested above; and
5. an executive or operating system which can attend to all the details of inputting, filing, allocating, loading, binding, sequencing, outputting, and the like which might be required.

SYNTACTIC ANALYSIS

Before discussing, in the next section, the descriptive language L_D in which the syntax and, to some extent, the semantics of the programming language L_P are provided, it is important to understand how the syntactic analysis is accomplished in our compiler model and how that portion of the language L_B program representing the syntactic analysis may be derived (mechanically) from syntax rules encoded in language L_D . For the purpose of this section we can think of the syntax language as BNF with a few notational changes (to make the representation of the analysis program in language L_B easier) plus a minor extension. The notational changes are as follows: for syntactic categories we will use simple identifiers, eliminating the [] brackets* normally used with BNF; the terminal symbols of the language will be singly quoted. The minor extension is to allow the declaration of those syntactic categories which are considered tokens—that is, recognized automatically by lexical analysis. An example including simple expressions over integers and an assignment statement is:

```
TOKENS (IDENT, INTEGER)
PRIMARY ::= IDENT | INTEGER
FACTOR ::= PRIMARY | '(' EXPR ')'
TERM ::= FACTOR | TERM '*' FACTOR
EXPR ::= TERM | EXPR '+' TERM
ASSG ::= IDENT '=' EXPR
```

Syntactic analysis will be accomplished by transforming syntax rules such as the above into a *reductions analysis program* in language L_B . We cannot give a transformation which will work for an arbitrary grammar; the technique which we will discuss depends upon the language being a precedence language. One might ask at this point why we do not utilize one of the standard "syntax driven" (predictive) analyzers to perform the syntactic analysis.⁸ Our reasons are, basically, that error recovery is much more easily handled with the scheme proposed here and that once a language has been verified as being a precedence language, we are certain that the language is not ambiguous. Our feeling is that programmers who are provided with facilities to extend the syntax of a language should also be given the measure of protection that guaranteed nonambiguity provides.

* [] have been used in printing this paper in place of the more conventional < >.

Precedence languages (as distinct from operator precedence languages⁹) were introduced by Wirth and Weber¹⁰ and include most of the current programming languages. The basic idea of a *simple precedence* language is a language such that each pair of syntactic categories and/or terminal symbols enjoy at most one of the three relations: = (equal), < (yielding), or > (taking) precedence. Any pair of symbols which appear adjacent in some syntactic rule have equal precedence; any terminal symbol appearing left adjacent to a syntactic category symbol in some rule yields precedence to all the symbols which can be leftmost symbols of any construction of that category, and so on. In the example above some of the precedence relations are:

```
('      = EXPR      = ')'
TERM    = '*'      = FACTOR
'*'    < PRIMARY
'*'    < '('
FACTOR > '*'
etc.
```

1 since PRIMARY and '(' may be the leftmost symbols of a FACTOR

2 since FACTOR may be the rightmost symbol of a TERM

Details are provided in Refs. 10 and 11.

Given a simple precedence language, we can parse a string (program) in that language, left to right, as follows: suppose that the given string (of terminal or token symbols) is I_1, I_2, \dots, I_M and that at some time during the parse we have reduced this string to

$$P_1 \dots P_N / I_j \dots I_M$$

where $P_1 \dots P_N$ are terminal or syntactic type symbols comprising the "parse stack" and $I_j \dots I_M$ are the input symbols which have not yet entered into the analysis (except for supplying right context). Then the next step depends upon whether $P_N > I_j$ on the one hand or $P_N = I_j$, or $P_N < I_j$ on the other (P_N not having any precedence relation with I_j indicated an error in the input string). That is, if $P_N > I_j$ then there exists a *phrase* $P_1 \dots P_N$ determined by $P_{i-1} < P_i = P_{i+1} = \dots = P_N$, and, by this, we mean that there exists some syntactic category R and syntax rule

$$R ::= P_1 \dots P_N$$

We "reduce" the string by

$$P_1 \dots P_{i-1} P_i \dots P_N / I_j \dots I_M \\ \rightarrow P_1 \dots P_{i-1} R / I_j \dots I_M \\ \text{or} \\ \dots P_1 \dots P_N / \dots \rightarrow \dots R / \dots$$

and proceed, comparing R and I_j . In the other case, $P_N < I_j$ or $P_N = I_j$, we effectively increment N and j by one and set $P_{N+1} = I_j$ —i.e. bring the next input symbol (I_j) into our "parsing stack"; the "reduction" can be depicted:

$$P_1 \dots P_N / I_j I_{j+1} \dots I_M \\ \rightarrow P_1 \dots P_N I_j / I_{j+1} \dots I_M \\ \text{or} \\ \dots P_N / I_j \dots \rightarrow \dots P_N I_j / \dots$$

A language which is not a simple precedence language may be a *higher order precedence* language. That is, a language in which, in general, the precedence relations hold, not between pairs of single symbols, but between strings of symbols, where the number of symbols in the left (right) string does not exceed some constant $m(n)$. For minimal m , n a language is said to be an (m,n) -precedence language ($(1,1)$ -precedence thus being equivalent to simple precedence). The parsing scheme sketched above also works for higher order precedence languages so long as the proper number of symbols are inspected.*

Now, let us think of the I 's and P 's as symbol descriptors. Each I is either a pointer to the terminal symbol table, or is a pointer to the literal or symbol table and contains the appropriate syntactic type code; each P is like an I or is a pointer to a computation and, again, carries the appropriate syntactic type code. Adding some machinery for disposing of the phrases which are encountered, we might have replacement rules (almost "statements" in language L_B) such as

$$\dots \text{TERM} / '*' \dots \rightarrow \dots \text{TERM} '*' / \\ \dots \text{EXPR} '+' \text{TERM} / \dots \\ \rightarrow \dots \text{EXPR\$EMIT} (\text{PLUS}, \text{EXPR}, \text{TERM}) / \dots$$

where $\text{EMIT} (\text{PLUS}, \text{EXPR}, \text{TERM})$ causes "output" of a PLUS pseudo-operation with two arguments, these being whatever descriptors were successfully typed EXPR, and TERM in the pattern, and returns as result a descriptor of the output (i.e.,

* Note that m,n are maxima for the entire language; for "most" symbol pairs it will probably be sufficient to inspect just those symbol pairs, and not strings of symbols of which the left (right) is rightmost (leftmost).

one having the *table* code of the "computation" table and *line* being where the PLUS was placed in the "current output area" of the computation table); the prefix EXPR\$ indicates that the result is to be syntactically typed (i.e., coded in its *type* field) EXPR.

It is a straightforward task to generate a set of such replacements from the syntax rules for a (generally higher order) precedence language. Usually (see Ref. 11 for details), we have one replacement corresponding to each alternate construction of each syntactic type, plus a replacement for each possible following symbol in order to "move" input into the parsing stack, all ordered appropriately to insure that reductions are not made prematurely. If we have the complete set of rules plus some mechanism to scan all the patterns to find the one applicable and then to make the corresponding replacement, we have a syntactic analyzer. However, it is better to introduce some control, allowing us to avoid looking at all patterns each "cycle", by means of the following techniques:

1. Grouping (and ordering) all replacements with the same symbol "on top of the parse stack" (i.e., immediately left of the /);
2. Attaching a label to the first replacement rule of each group.
3. Introducing the following "actions" which can follow a replacement rule: TRY (label) which indicates that the pattern indicated is to be tried next; ERROR which announces an error, and EXIT, which terminates the process.
4. Following each rule with TRY(t) where t is the label corresponding to the new top of parse stack (resulting from the replacement), or with ERROR or with EXIT if an error condition maintains or the "largest syntactic type" has been recognized.

Now presuming a control mechanism which, "pointing" to some replacement, tries the pattern and, upon success, performs the replacement and the action(s) following and, upon failure, tries the next pattern, we have a language L_B program for syntactic analysis. The program for the simple example given above is (where we have taken a few liberties to show the kind of program our language L_D to L_B translator would produce):

IDENT.T	... IDENT / '=' ...	→ ... IDENT '=' / ...	TRY(OP.T)
	... IDENT / ...	→ ... PRIMARY \$ IDENT / ...	TRY(PRIMARY.T)
INTEGER.T	... INTEGER / ...	→ ... PRIMARY \$ INTEGER / ...	TRY(PRIMARY.T)
PRIMARY.T	... PRIMARY / ...	→ ... FACTOR \$ PRIMARY / ...	TRY(FACTOR.T)
RPAR.T	'(EXPR)' / ...	→ ... FACTOR \$ EXPR / ...	TRY(FACTOR.T)
		→	ERROR
FACTOR.T	... TERM '*' FACTOR / ...	→ ... TERM \$ EMIT(TIMES, TERM, FACTOR) / ...	TRY(TERM.T)
	... FACTOR / ...	→ ... TERM \$ FACTOR / ...	TRY(TERM.T)
TERM.T	... TERM / '*' ...	→ ... TERM '*' / ...	TRY(OP.T)
	... EXPR '+' TERM / ...	→ ... EXPR \$ EMIT(PLUS, EXPR, TERM) / ...	TRY(EXPR.T)
	... TERM / ...	→ ... EXPR \$ TERM / ...	TRY(EXPR.T)
EXPR.T	... EXPR / ')' ...	→ ... EXPR ')' / ...	TRY(RPAR.T)
	... EXPR / '+' ...	→ ... EXPR '+' / ...	TRY(OP.T)
	... IDENT '=' EXPR / ...	→ ... ASSG \$ EMIT(STORE, IDENT, EXPR) / ...	EXIT
		→	ERROR
OP.T	... / IDENT ...	→ ... IDENT / ...	TRY(IDENT.T)
	... / INTEGER ...	→ ... INTEGER / ...	TRY(INTEGER.T)
	... / '('	→ ... '(' / ...	TRY(OP.T)
		→	ERROR

SPECIFICATION OF SYNTAX AND SEMANTICS VIA LANGUAGE L_D

The BNF-like syntax language introduced in the previous section is our starting point for language L_D . That is, language L_D admits data descriptions of the various data (tables and the like), declaration of pseudo-operations, and so on to be included in the compiler; allows specification of the syntax of language L_P in the above notation; and has provisions for the specification of "semantics", the various actions which are to be taken whenever a certain syntactic construction is found (i.e., the reduction is performed). The L_B program which results from translation of an L_D program contains the L_D data declarations intact plus a reductions analysis program within which the "semantics" have been imbedded for analysis of language L_P program text. We will describe L_D in this section by presuming a "basic" L_D which allows specification of syntax rules in a manner similar to that sketched in the previous section and assume it has provisions for handling data descriptions and, given this basic L_D , we will motivate and describe a number of "extensions".

The first extension is to allow the attachment of "semantics" or "interpretation of the parse" to each syntactic construction. This is accomplished by adjoining to a construction a bracketed "interpretation"

or "output specification". Using the previous example we might write:

```
TOKENS (IDENT, INTEGER)
PRIMARY ::= IDENT |
          INTEGER
FACTOR  ::= PRIMARY |
          '(EXPR)' [EXPR]
TERM    ::= FACTOR |
          TERM '*' FACTOR
          [EMIT(TIMES, TERM,
                FACTOR)]
EXPR    ::= TERM |
          EXPR '+' TERM
          [EMIT(PLUS, EXPR,
                TERM)]
ASSG    ::= IDENT '=' EXPR
          [EMIT(STORE, IDENT,
                EXPR)]
```

The L_D to L_B translator will arrange for the appropriate prefixing (e.g., $EXPR\$EMIT(PLUS, EXPR, TERM)$) and "type promotion" when no bracketed interpretation is given (e.g., $\dots PRIMARY / \dots \rightarrow \dots FACTOR\$PRIMARY / \dots$). We note here that it is not necessary that pseudo-code output be specified; an L_D description which would output a "syntax tree" is given by:

```

TOKENS (IDENT, INTEGER)
PRIMARY ::= IDENT [EMIT(IDENT)] |
          INTEGER
          [EMIT(INTEGER)]
FACTOR  ::= PRIMARY
          [EMIT(PRIMARY)] |
          ('EXPR') [EMIT('(',
                    EXPR, ')')]
TERM    ::= FACTOR
          [EMIT(FACTOR)] |
          TERM '*' FACTOR
          [EMIT(TERM, '*',
                    FACTOR)] and so on.

```

We now proceed to introduce another extension. Matching some syntactic construction is not always enough; that is, there may exist identical constructions of several different syntactic types (e.g. [identifier] in ALGOL). We thus allow a *predicate* to be attached to a syntactic construction and arrange that both the matching of the pattern (i.e., recognition of that construction) and the truth of the predicate are required in order that the corresponding reduction be made. A predicate is any Boolean expression in language L_B which involves "and", "or", and "not" combinations of relationals over the integers, fields of tables, and working variables which have been declared. As an example we might have

```

REALVAR ::= IDENT WHEN DATATYPE
          (IDENT) EQ REALTYPE;
INTVAR  ::= IDENT WHEN DATATYPE
          (IDENT) EQ INTTYPE;
and so on.

```

Here IDENT in the context DATATYPE-(IDENT) is understood to mean the *line* field (index into the symbol table) of the descriptor syntactically typed IDENT; and DATATYPE () has presumably been previously declared as a field of the symbol table.*

The next extension is to augment the language by allowing "computations" (in language L_B) to be attached to a construction—computations to be carried

* It should be noted that we are not intending to use this predicate mechanism to discriminate various data types—we are here assuming that lexical analysis assigns the appropriate data (i.e., syntactic) type automatically from information in the symbol table. We also note that the division of the testing into a determination of structure followed by application of predicates to certain structural elements is similar to the scheme employed by Fenichel in the FAMOUS System.^{1,2}

out as the corresponding reduction is performed. Again for example, we might have

```

DECL ::= 'REAL' IDENT DO DATATYPE
        (IDENT) = REALTYPE; |
        'INTEGER' IDENT DO DATA-
        TYPE(IDENT) = INTTYPE;

```

and so on.

Another extension, this time to the way in which we can write a syntax rule, will help alleviate some of the trouble and confusion which recursive syntactic constructions sometimes introduce. Consider for example, the two rules

```

EXPR ::= EXPR | EXPR '+' TERM
ARGLIST ::= EXPR | ARGLIST ',' EXPR

```

The first construction may be entirely proper and desirable; that is, it specifies the left associativity of the addition operator as well as indicating that '+' has lower "binding power" than the operators (presumably '*' etc.) used to make TERMS. However, the second construction is generally confusing in that the expressions making up an argument list are "equal" = i.e., have neither left nor right grouping. A preferable syntactic form would be

```

ARGLIST ::= EXPR {' EXPR}

```

where { . . . } is interpreted "choose zero, one, two, etc. occurrences of . . .". We would like the *result* of recognition of this construction to be a *list* of the EXPRs recognized with which we can transact further. We extend L_D by admitting such constructions as

```

ARGLIST ::= EXPR {' EXPR} [EMIT
                    (LIST, EXPR)]

```

where LIST is a defined pseudo-operation and EXPR in the context EMIT(LIST, EXPR) denotes a (first-in-first-out) queue of the descriptors successively recognized as EXPRs. Language L_B has facilities for dealing with such lists: counting elements, accessing individual or groups of elements, and so on.

As another example we might have (for reasons which will become clear when we discuss computational macros)

```

REF ::= IDENT [EMIT(IDENT)] |
        IDENT ('EXPR {' EXPR}')
        [EMIT(IDENT, EXPR)]

```

Finally, we require an extension which will help us to recover from errors detected by the syntactic

analysis. We note that the language L_D to L_B translator will supply rules of the form

→ ERROR

as the last rule of a group which may not otherwise contain a rule guaranteed to be successful. The extension is to admit statements of the form

OTHERWISE (comp) stuff

where “comp” is any syntactic type or terminal and “stuff” contains interpretations, predicates, or computations similar to any correct construction; the L_D to L_B translator will insure that the “stuff” is done rather than an error being signalled. An example:

```

    OTHERWISE (EXPR) DO BEGIN
      TRY(EXPR.R)
    EXPR.R      ... '(' EXPR / ...
→ ... '(' EXPR' / ... TRY(RPAR.T)
→ PRINT (“INCORRECT OPERATOR FOLLOWING EXPRESSION”);
    TRY(RECOVER) END
  
```

what we are trying to suggest here are several things, to wit:

1. L_B has a facility for printing (error) comments;
2. replacement rules can be directly programmed as “computations” in L_B ;
3. a series of “computations” can be bracketed by BEGIN END brackets; etc.

A stronger way of putting this is that our “operational expectation” is that much of the syntax and semantics of an L_P can be provided via L_D and result automatically in an L_B program for syntactic analysis and interpretation of the parse. However, since the “final” program is in L_B we are free to modify, extend, etc. the L_B program to account for syntactic vagaries (non-precedence situations), error recovery, and the like either by modifying the L_B program resulting from L_D translation or by inserting this kind of thing into L_D by the device sketched above. Thus one could have a more-or-less “pure” L_D description of a language available to a user as a reference document; the details of error recovery, handling of special cases, and so on might be represented only in a modified version of the L_B program resulting from translation of the L_D program, and supplied only to “experts”. The point is that both declarative and imperative versions of the syntactic

analysis and parse interpretation are available in a similar form, surely an aid to documentation.

ADDITION OF DEFINITIONAL OR “MACRO” FACILITIES TO A LANGUAGE

With the compiler model and other machinery outlined above in mind, we now turn to the question of imbedding the macro definition and call facilities into a given language L_P . We will organize our discussion around the time at which the macro facilities are employed and thus discuss

Text Macros—which are employed prior to lexical analysis;

Syntactic Macros—which are called during syntactic analysis; and

Computational Macros—which are called subsequent to syntactic analysis.

We now proceed to a discussion of each of these three types.

Text Macros

By text macros we mean macro facilities which, from the point of view of our compiler model, are contained in the interface which supplies program text to the lexical analyzer. Indeed, one might take either TRAC¹ or GPM² (they are very similar devices) as perfectly satisfactory pre-lexical macro devices essentially as they stand. In any event, since these kinds of macros are really outside the scope of the compiler model we have described and, particularly, since two excellent macro systems which are sufficient for this purpose have been described in the literature, we shall not discuss this type of macro further.

Syntactic Macros

What we have in mind here is a facility which allows one to define what are in effect new syntactic structures in terms of the given syntactic structures of the language L_P and previously defined “syntactic macros”. We require three things: a syntactic augment to L_P to allow the definition and call of syntactic macros; a mechanism, within our syntactic analyzer, for “recording” the definition of the syntactic macros; and, a mechanism within our syntactic analyzer (and lexical analyzer) to handle the call of these macros.

The definition of syntactic macros will equate a "macro form" which is some syntactic structure—a string of terminal and token symbols mixed with identifiers, previously declared as formal parameters bearing a given syntactic type—with a "defining string" which is a string of characters intermixed with the formal parameter names. Upon recognition of the macro form, the syntactic analyzer will submit the corresponding defining string to the lexical analyzer after placing the actual parameters occurring in the macro call on an argument list. The lexical analyzer will then lexically analyze the defining string in a normal fashion with the exception that occurrences of the formal parameter names (as identifier tokens) will result in the output of the descriptor corresponding to the actual parameter (already processed and stored on the argument list) rather than a descriptor for that identifier. This lexical output will precede the normal lexical output (or previously arranged macro output, if the macros are defined recursively). Note that we presume that the lexical analyzer deals only with a single descriptor of the actual parameter, however complex (syntactically) it might have been in the macro call, and merely substitutes this descriptor for the occurrence of the corresponding formal parameter in the defining string, nothing more. The computation tree or whatever resulted from the analysis of the actual parameter within the macro call is not touched.

We will assume that syntactic macro definitions occur at the head of a program (or of a block, if you want), since the syntactic analyzer may be considerably modified (re-constructed) by such definitions and it might be prudent to restrict such operations to the beginning of a compiler run. (Clearly, generally useful syntactic macro extensions to a compiler could result in the "extended" compiler being filed under some new name to save the overhead of re-constructing the compiler "on the fly" each time it is called.)

Let us now propose an extension of the syntax of L_P to allow definition and call of two varieties of syntactic macros. (Each particular L_P would doubtless be handled somewhat differently; the intention here is only to propose something we can make reference to in the sequel). For this we will use the BNF notation:

```
[s-macro-parameter-decl] ::=LET [identifier]
BE [syntactic type]
[s-macro-definition] ::=MACRO [macro form]
MEANS '[defining string]' |
```

```
SMACRO [macro form] AS [syntactic type]
MEANS '[defining string]'
```

The [s-macro-parameter-decl] declares the [identifier] to be a formal parameter which may subsequently appear in a [macro form] and its associated [defining string]; further, the [identifier] is declared to be of the given syntactic type. The [syntactic type] may be either a syntactic type (identifier) already defined for L_P or may be a new syntactic type.*

The [macro form] may be any sequence of token and terminal symbols for L_P and will generally include instances of certain of the declared formal parameters ([identifier]s). The assumptions about lexical handling are: (1) it is presumed that in lexically analyzing the [macro form] the lexical analyzer will encounter and "normally" treat the tokens and terminals; (2) any identifier token which has not been declared as a formal parameter will, upon processing of the [s-macro-definition] be so tagged that lexical analysis will subsequently treat that identifier as a terminal symbol (i.e., it will have residence in the symbol table so typed that subsequent lexical analysis of it will produce a descriptor indicating not symbol table but terminal symbol table—with "new lines" in that table generated for such "defined new terminals" as necessary); and, (3) the [defining string] will be stored away as a string of characters and the macro form will, in effect, have an associated pointer to this string (i.e., we are assuming that the quotes protect the [defining string] from any lexical analysis).

The calls of the two types of macros (MACRO and SMACRO) are different. A call of a MACRO type macro will be of the form

```
% [macro call]
```

where % is a special character presumed not heretofore used in L_P and [macro call] is an instance of some MACRO defined [macro form] where the formal parameter places have actual parameters which are syntactic units of the declared type. The successful processing of the [macro form] will result in the corresponding [defining string] being submitted

* The system will have to insure that an identifier in this context is "looked up" in a special table of syntactic types (or is handled by some equivalent mechanism) and is not treated as an [identifier] in L_P in order to avoid any conflict of the (possibly mostly unknown to the user) identifiers used in L_D to name syntactic types of L_P ; further there will be a provision for introducing new syntactic types. One possible scheme is to use special delimiters to distinguish syntactic types from other identifiers; for example: .EXPR. .NEWTYP. etc.

to the lexical analyzer and the actual parameters with their associated names being placed on an argument list. The % [macro call] "phrase" will then be completely wiped off the parse stack and syntactic analysis will continue. Note that % is a special terminal which takes precedence over any symbol which might appear immediately to the left of it (to insure that the macro is expanded to provide the proper right context for the symbol preceding the % in the L_P text). Example

LET N BE INTEGER
 MACRO MATRIX (N) MEANS 'ARRAY(1:N, 1:N)' is essentially equivalent to a syntactic definition of the form

MACRO::='%' 'MATRIX' ('INTEGER')

where the macro syntactic type is handled differently from other syntactic types in that % takes precedence over everything and recognition of the phrase "% ..." causes elimination of the phrase (after "interpretation") rather than reduction to a single descriptor typed MACRO.

A call of the macro of the form

% MATRIX (25)

in most contexts will result in the equivalent of having written:

ARRAY (1:25, 1:25)

in that context.

Similarly

MACRO A MEANS '+B*C+'

followed by

X = Y %A Z

will result in the equivalent of having written

X = Y+B*C+Z

The definition and use of the MACRO macros must be done with some care. In particular, the right-most component of the [macro form] must be carefully chosen so that the complete [macro form] will be recognized as a phrase. For example, the right-most component could be a terminal symbol in L_P which will take precedence over the symbol following the macro call in all contexts in which it might appear. The use of an identifier which is treated as a new terminal symbol as right-most component would eliminate any trouble, since its precedence will automatically be taken as greater than any

symbol which may succeed it. Also, one could introduce a special right delimiter (matching the % left delimiter) and insist on both left and right delimitation of the MACRO macro call.

We note that the text macros and the MACRO macros are similar in that both require some sort of trip character to announce their call. Indeed, in many situations the choice of one type over the other might be quite arbitrary. However, there are situations in which the MACRO macros are clearly preferable. One situation is when the verification of the correct syntactic type of the actual parameter is desired prior to the expansion of the macro, for example to allow better control in announcing to the user the exact circumstance in which an error occurred. We note also that the form of macro call for text macros would very likely be quite rigid (e.g., #(name, arg, ..., arg) in TRAC and § name, arg, ..., arg; in GPM) while the form of MACRO macro calls is not restricted except for the position of the trip character, %.

There are many situations in which neither the text nor the MACRO macros are adequate. In particular, the requirement for a trip character in the call required to keep the syntactic analysis from going askew may be distasteful in many cases. The SMACRO form of syntactic macro differs from the MACRO form in that a syntactic type of the *macro form* is provided (AS[syntactic type]) and thus the call of an SMACRO can be accomplished without using the special % character. While the declaration of the MACRO and SMACRO macros and the handling of their calls, once they have been recognized, are very similar, the actual recognition of the two kinds of macros is quite different. That is, the MACRO macro call has the % character to trigger recognitions while with the SMACRO macro calls the handling of the recognition is just as though the macro form had been declared (in L_D) as another construction of the given syntactic type and thus enjoys all the attendant advantages and disadvantages. As an example we can write the first example above as:

LET N BE INTEGER
 SMACRO MATRIX(N) AS ATTRIBUTE
 MEANS'ARRAY(1:N, 1:N)'

and call the macro via, for example:

...MATRIX(25) WALDO...

so long as the calling string is in an allowed context for the (presumed previously defined for L_P) syn-

tactic type ATTRIBUTE. The macro definition in L_D of the form

ATTRIBUTE ::= 'MATRIX' ('INTEGER')

and accompanying interpretation or semantics accomplishing whatever ARRAY(...) accomplishes.

As another example of the SMACRO form, suppose that L_P has arithmetic expressions (EXPR), relation operators (RELOP), and relations (RELATION) as syntactic types, and also allow ANDing and ORing of relations, where the syntax for RELATION was given as

RELATION ::= EXPR RELOP EXPR

with appropriate accompanying interpretation. By the next recognition of the macro form the parse stack would be

...	O	<	B	<	O	/	...
	↓				↓		
	PLUS A = 1				PLUS C = 1		

and so on.

It is clear that some measure of care must be taken in the use of syntactic macros and it is implicitly assumed here that this kind of tool would probably be placed only in the hands of a reasonably sophisticated user. We do, however, submit that the methods incorporated here for syntactic analysis would allow not only "catching" but more-or-less reasonably useful "commenting" upon violations to the given syntax (via L_D) by poor use of the SMACRO facility.

By virtue of the particular technique which we propose to utilize for syntactic analysis it appears possible, by saving an appropriately encoded representation of the syntax rules and the precedence relations for L_P , to handle syntactic macros by "incremental compiler changes" if the representation of the L_B program which "runs" the reductions analysis is as appropriately organized coding for an interpretive system.

It would, in principle, be possible to allow the attachment of more "semantics" to a [macro form] than just the [defining string]. That is, language L_D (or L_B) fragments appropriately delimited could presumably be handled along with or in place of the [defining string] by utilizing the presumed facilities of the compiling system to reconstruct the compiler even while it is processing L_P programs containing L_B statements. However, this approach seems practically infeasible and we are led to propose some

"built in macros" to accomplish this function for some of the more "practical" cases; we will discuss one of these and mention a few others to give the flavor.

There is one particular facility which would be extremely useful and which cannot be handled by the syntactic macros (except by some real trickery in defining L_P originally) as defined above, namely the introduction of new data types. The problem is that these would necessitate the ability to make appropriate symbol table (type field) entries. We thus propose the following "built in" macro, called via

% DEFINETYPE ([identifier], [syntactic type])

which associates the given syntactic type code (very likely a new syntactic type rather than one of the syntactic type given for L_P) with the [identifier] in the symbol table.

Given this, plus the ability to "talk about" these type codes via

LET [identifier] BE [syntactic type]
and

SMACRO [macro form] AS [syntactic type]
MEANS [defining string]

we can then deal with definition and manipulation of new data types. Other built in macros would include some allowing the augmenting of the symbol table by new fields and subsequent reference to those fields, access to and provision for manipulation of those fields of the symbol tables (or other tables) which contain data allocation information, and so on. Ref. 13 describes an extensible version of ALGOL-60 based on the ideas in this paper and contains several elaborate examples of the use of the various macro facilities.

Computational Macros

The idea of computational macros is quite straightforward and their use within a compiler is quite cheap (as compared with the manipulations required to handle syntactic macros). The idea is this: we may optionally associate with any identifier a sequence of pseudo-instructions (actually a computation tree) which contain, in general, some formal parameters as arguments. The completed processing of the "reference" (i.e., the identifier and all the expressions comprising its actual argument list) can then be followed by the macro expansion, i.e. the replacement of the reference by a copy of the computation tree with each formal parameter argument replaced by the corresponding (completely processed) actual pa-

parameter. The computation tree, or macro skeleton, would be defined by associating with the macro name (the identifier whose occurrence in a reference will trigger the macro expansion) the result of syntactically analyzing and interpreting an expression * which is the "defining string" for the macro. Note here that one difference between the syntactic and the computational macros is that the defining string for syntactic macros is not touched until a macro call, at which time it looks (to within parameter substitution) like raw input; on the other hand, the defining string for a computational macro is analyzed and interpreted into a series of pseudo-instructions on first encounter.

Again, we require three things to handle such macros: a syntactic augment to L_P to allow the definition of the macros; a mechanism within the syntactic analyzer for "recording" the definition; and, a mechanism for recognizing and effecting the call of the macro. In the above discussion of the presumed symbol table contents and the handling of references we left some connections for handling these macros. First, let us propose the syntactic augments to L_P , to wit:

```
[c-macro-parameter-dec] ::= TAKE [identifier]
                          {AS [syntactic type]};
[c-macro-definition] ::= MAP
  [identifier] {[parameter] {[parameter]}} =
  [expression] {UPON [syntactic type]};
```

Leaving out the optional AS and UPON parts for the moment, we assume that the "TAKE [identifier];" declares the identifier to be a formal parameter for subsequent definition of c-macros. The identifier in the MAP is assumed to be a variable (optionally subscripted) which is declared in the L_P program; the [parameter]s are formal parameters. (We could, of course, eliminate the actual declaration of the formal parameters and let their occurrence in the MAP definition indicate their formal parameter-hood; subsequent remarks on the optional AS syntactic type will show why we have not chosen to do this.) The expression will be presumed to be any (arithmetic or other) expression allowable in L_P and will generally contain occurrences of the formal parameters. The mechanisms are then roughly as follows: The [c-macro definition] causes the expression to be syntactically analyzed and interpreted, resulting

* Or, more generally, a complete procedure so long as the procedure returns a value and the code selection phase of the compiler can cope with full procedure calls where a variable normally appears.

in a sequence of pseudo-instructions; the only augment required to the syntactic analyzer is to handle the formal parameter identifiers occurring by replacing each with some appropriate descriptor,† a completely trivial matter. Further, the symbol table entry for the identifier is set to indicate that it names a macro and a pointer is inserted to point to the computation tree produced for the expression. Assume that any reference say [identifier] {[,expr] {[,expr]}} is then syntactically analyzed to produce a "pseudo-instruction" where the [identifier] descriptor is the "pseudo-operation" and the [expr] descriptors follow it as "arguments".

Then, noting from the symbol table entry for the [identifier] that it is a macro, the (macro skeleton) computation pointed to by this entry is copied with appropriate replacement of formal parameter descriptors by the actual parameters in the manner suggested above. In the event that references occur within the mapping computation, the mechanism will recursively expand these, and so on.

As an example, suppose that A, B, G are declared as 2, 1, 0 dimensional arrays; then a computational macro for the "mapping functions" for B and A might be given by:

```
TAKE I;
TAKE J;
MAP B(J) = J * 2 + G;
MAP A(I,J) = 25*I + B(J+1);
```

The result of processing these might result in symbol table and descriptor table entries as follows:

Identifier	Map Type	Macro Pointer
G	Normal	
B	Macro	4
A	Macro	15
1: TIMES	$\pi_1 = 2$	
4: PLUS	① G	
7: TIMES	$= 25 \pi_1$	
10: PLUS	$\pi_2 = 1$	
13: B	⑩	
15: PLUS	⑦ ⑬	

Here π_j stands for a descriptor of formal parameter j , circled integers refer to previous pseudo-instructions, $= 25$, etc. indicates the literal 25, etc. A reference to $A(X+Y,Z)$ then might be parsed to yield:

```
100: PLUS X Y
103: A ⑩⑩ Z
```

† e.g. presume a "formal parameter table" (without content) and use *table* code to indicate formal parameter-hood and *line* field to indicate which one.

and then expanded to yield:

```

106: PLUS (109) (122)
109: TIMES = 25 (100)
112: PLUS Z = 1
115: PLUS (118) G
118: TIMES (112) = 2
122: LOCATE B dB (115)
126: LOCATE A dA (106)

```

Here the LOCATE pseudo-instructions indicate a reference to the data element indicated by the first argument and where the constant and variable parts of the displacement from the base of the area containing the values for that data element are given by the second and third arguments.

As another example, consider the following mapping function which provides contiguous storage for a 4×4 (upper left) triangular matrix $M()$:

```

TAKE I;
TAKE J;
MAP M(I,J) = (11-I) * 1/2 + J - 6;

```

This example raises the whole set of issues to do with storage allocation (presumably the declaration of M as a 4×4 array would have resulted in 16 words being reserved for it rather than the 10 we desire), providing, filing, and subsequently obtaining and incorporating into the compiler the description of "global" data, and so on. While we cannot go into these issues here, Ref. 14 describes a rather elaborate data declaration facility which we have added to a subset of PL/I, using the computational macro techniques described above, and Ref. 13 discusses the handling of such allocation within the framework of ALGOL-60.

As a final example which illustrates the use of computational macros for other than data mapping applications, consider the following:

```

TAKE X AS EXPR;
MAP F(X) = SIN(X) + COS(X);

```

Now let us touch briefly upon the use of syntactic types in declaring the formal parameters for computational macros and in defining the macros themselves. First, unless the formal parameters are explicitly syntactically typed we will assume that they are [integer-expression]s (or the equivalent). This will allow proper syntactic analysis of mapping expressions, generally the most usual use of computational macros. In general, however, it is clear that syntactic type must be given in order that the expres-

sion (which may be a Boolean or string, etc. expression) can be parsed. As to the use of the UPON [syntactic type] facility, we will assume that any given identifier may have a number of interpretations (e.g. as a floating as well as an integer valued variable or as a label as well as an integer array name) and there will be a symbol table entry for each interpretation (MAP) with the corresponding "mapping release" syntactic type recorded. A unique interpretation is chosen on the basis of the syntactic type of the reference by matching this "mapping release" syntactic type.

If the control of when references are "expanded" is vested in some L_B action which is called at the statement or some other level, then a computation tree can be assembled and the mapping expansion of the whole tree "fired off" allowing the mapping release type to be matched not only against the syntactic type of the reference but against those of its ancestors, thus allowing reasonably fine contextual control of different mappings for the same identifier. The stack example in Reference 13 illuminates this notion.

There is another issue which should be mentioned but which will not be pursued very far in the present paper. This is the issue of argument types and numbers of arguments expected of references as they relate to the question of selection of proper mapping (i.e., interpretation). We will presume that the symbol table entry for an identifier, expected to appear with arguments, contains a description of each of these arguments. It would be desirable (it is actually done in the system implementing the language described in Ref. 14) to have an argument matching involved in the searching and decision making to select the "proper" interpretation of a given reference. It is possible to envision a table of actual argument type versus desired argument type containing "degradation factors" (per identifier, per block, per program, or for all the time) to be applied to the measure of desirability of each interpretation of a given reference. Such a table, in conjunction with the mapping release syntactic type, would provide a quite sophisticated criterion for the selection of one from a number of possible interpretations, as well as allow automatic specification of argument conversions, and so on.

It should be remarked that to properly handle such schemes it is desirable to introduce the idea of *syntactic classes* into the syntax language to allow "grouping" of syntax categories without change of

syntactic type. For example, using $::=$ to denote a syntactic type as, really, a class, we would interpret

```
EXPR ::= INTEGEREXPR | REALEXP  
      | STRINGEXPR | etc.  
ARGLIST ::= EXPR {',' EXPR}
```

as allowing an ARGLIST of different kinds of expressions each retaining their peculiar syntactic types, in order that subsequent inspection of the elements of the list as arguments would have the "proper" syntactic type for inspection (to save descending into the computation tree to "dig out" the real type).

CONCLUSION

Let us briefly reiterate the point of view which this paper has developed. We have proposed a compiler model, a compiling system, and two programming languages, L_D and L_B , with which a particular compiler should be more-or-less readily constructable and which result in a program whose efficiency is determined largely by the trouble we choose to take in providing an optimized translation* of language L_B . We have further suggested three types of definitional facilities which will provide the user with a kit of tools to extend a given language to more closely mirror the natural means of exposition in the application area in which he is involved. We want particularly to emphasize that extensions to the user's language need not be based entirely on exploitation of the macro facilities. Indeed, we have left two handles: the ability to add new built in macros and the ability to fall back to language L_D and rewrite (some portion of) the compiler. Given that the pseudo-operations available provide adequate primitives for representing the extensions desired (and that the optimization and code selection facilities are adequate) this latter facility should be useable by users—not just by compiling system buffs.

As to the status of the various languages, translators, and other facilities discussed in this paper, we do not, at this time, have in hand a complete system with a compiler for some user programming language augmented by all three kinds of macro facili-

* Clearly, the development of more and more optimal translators for L_B given, initially, a reasonably straight forward translation into "coding" for an interpretive system, is a task which can proceed in parallel with the development of compilers via programs in L_D and L_B . The effect on the user should be no more than the fact that his compilations may go faster.

ties. On the other hand, these proposals are not merely represented as flights of our fancy, to wit:

1. A compiling system called TRANGEN, and a realization of L_B , called TRANDIR, have existed for some years (and gone through five major language changes and system implementations).^{6,7} The TRANDIR/TRANGEN facilities have been used for a number of compilers (PL/1, ALGOL, TRANDIR, FORTRAN IV, a data description language, etc.) on a number of computers (IBM-7094, CDC-1604, GE-635, M-490, etc.)
2. A ("reference" language) version of language L_D has existed for some time¹¹ and has been used to translate, by hand, a specification of a language L_D description of an L_P into a reductions analysis program in TRANDIR.
3. Text macro facilities are currently available (but not necessarily "hooked in" to a compiler).^{1,2}
4. Computational macros have been utilized in a number of compilers; indeed we have developed a number of data description languages based on this idea.¹⁴

Finally, we ought to remark very briefly on the relation of the SMACRO macro facilities proposed herein to the "definitions in ALGOL" proposed by Perlis and Galler⁵ since their proposal is, to our knowledge, the only scheme of any generality described in the literature for syntactic macros. Our two approaches would appear to be reasonably similar. The main differences seem to be three: first, they restrict the syntactic categories which will admit extension to [block head], [type], [assignment statement], [arithmetic expression], and [Boolean expression], while we have not adopted such restrictions; secondly, their [macro forms] (our terminology) are more powerful than those proposed here in that they can accommodate multiple instances of the formal parameters in the [macro form] (meaning multiple occurrences of exactly the same structure in the macro call) because they use a "tree matching" scheme for recognizing the macro call while we use matching of (effectively linear) syntactic constructions; thirdly, our [macro form] is more powerful than theirs in that we allow the use of arbitrary syntactic types as parameters of [macro forms] and use

these in recognizing a macro call, while they do not ascribe syntactic type to their formal parameters.

In conclusion, it is our very strong feeling that languages with powerful definitional facilities must be placed in the hands of users. Indeed, the appropriate representation—something natural to the individual—of a problem (or solution) has more to do with the issue of solving problems than merely providing a nicety. To those who question this position, I suggest the following problem in long division:

XLVI		MCXXIV
------	--	--------

REFERENCES

1. C. N. Mooers, "TRAC, A Procedure Describing Language for the Reactive Typewriter," *Communications of the ACM*, March, 1966.
2. C. Strachey, "A General Purpose Macrogenerator," *The Computer Journal*, October, 1965.
3. C. J. Shaw, "A Specification of JOVIAL," *Communications of the ACM*, December, 1963.
4. "IBM Operating System/360, PL/1 Language Specifications," IBM Systems Reference Library, Form C28-6571-2, January, 1966.
5. B. A. Galler and A. J. Perlis, "A Proposal for Definitions in ALGOL," to appear in *Communications of the ACM*.
6. R. Bolduc, et al, "Preliminary Description of the TGS-II System," Document CA-6408-0111, Computer Associates, Inc., September, 1964.
7. T. E. Cheatham, Jr., "The TGS-II Translator Generator System," *Proceedings of the IFIP Congress, 1965*, Spartan Books, Washington, D.C., 1965.
8. T. E. Cheatham, Jr., and K. Sattley, "Syntax Directed Compiling," *AFIPS, volume 25, SJCC*, Spartan Books, Washington, D.C., 1964, pp. 31-57.
9. R. W. Floyd, "Syntactic Analysis and Operator Precedence," *Journal of the ACM*, July, 1963.
10. N. Wirth and H. Weber, "EULER: A Generalization of ALGOL and its Formed Definition: Part I," *Communications of the ACM*, January, 1966.
11. T. E. Cheatham, Jr., "The Theory and Construction of Compilers," (Notes for AM 219R, Harvard University, Spring, 1966) Document CA-6606-0111, Computer Associates, Inc., June, 1966.
12. R. R. Fenichel, "An On-Line System for Algebraic Manipulation," Ph.D. Dissertation, Harvard University, July, 1966.
13. T. E. Cheatham, Jr., "ALGOL-E, An Extensible Version of ALGOL-60," in preparation.
14. C. H. Christensen and R. W. Mitchell, "Reference Manual for the NICOL 2 Programming Language," First Edition, Computer Associates, Inc., in preparation.

FOUNDATIONS OF THE CASE FOR NATURAL-LANGUAGE PROGRAMMING

Mark Halpern

*Lockheed Palo Alto Research Laboratory
Palo Alto, California*

Few things worth saying can at once be said clearly; and whereof one cannot yet speak clearly, thereof one must practise speaking.

—Wittgenstein *renovated*

INTRODUCTION

A running debate, mostly subterranean, has long been going on over the suitability of natural language for use as a programming language. From time to time the debate surfaces in the form of sharp exchanges at technical conferences and strong letters to the journals, but these casual encounters have been insufficient even to make clear to the general reader what the issues are, let alone to resolve them. The absence of open and lively debate between those who favor and those who oppose natural-language programming has left the problem to be dealt with by each language designer as best he can, without benefit of others' experience and ideas. Two opposite but equally undesirable ways of handling the conflict are in common use: one is a mutual turning of backs by the two parties, as may be seen in the increasingly wide gulf between research and practice in the design of programming languages; the other is a tendency toward superficial, makeshift compromise, with the usual result of satisfying no one. The possibility that the issues underlying the controversy

are too fundamental to allow any useful exchange between the two parties cannot be dismissed, but it seems worth some effort to find out; the result should be at least a clearer idea of what we are disagreeing about.

The root issue may be put roughly this way: one school believes that a programming language need not and should not have its form dictated by the fact that it is in some sense addressed to a machine, but should be very close to the language its intended users ordinarily employ in their work, apart from the computer. The other school believes that the fact that a programming language is addressed to a machine is the inescapably decisive force in determining its shape, and that such a language will almost certainly be quite different from those in use between man and man. The first or "natural-language" school is often considered to be advocating the use of plain English as a programming language. This characterization, true so far as it goes, is a simplification of their position that is liable to serious misinterpretation; it may easily be taken to mean, for example, advocacy of languages like COBOL, which it does not. The second or "calculus" school (as we shall call them) see programming languages as needing a massive infusion of the rigor, precision, and economy exhibited by mathematical notation. Its members often suggest that a language

with these qualities would amply repay its users for their trouble even if it were not implemented on a computer, because it would give them for the first time a proper representation of their procedures.*

There is no quarrel between the two schools over how to call for numerical computation; they agree that the algebraic notation commonly offered by compilers is right for that purpose. Their agreement on this point, however, represents no new insight on either side, but only a happy coincidence of prejudices: in mathematical (and symbolic-logical) notation we have the one language that is both a natural language, in the sense defined above, and a calculus. The dispute over almost all other computer applications continues. In this dispute the writer makes no pretense to neutrality; this paper, as its title indicates, is an attempt to lay the foundation for a case in favor of natural-language programming. In doing so, however, we will have almost as much occasion to take issue with others favoring that cause as with those opposing it, for we of the natural-language party have been remiss in developing and presenting our ideas, and have been guilty of perpetuating a number of errors.

THE PROBLEM—WHAT IT IS AND WHAT IT ISN'T

It is of the utmost importance to correctly identify the problem that natural-language programming is being proposed to deal with. A number of the proposal's critics have pointed out that by far the greater part of the effort that goes into programming lies in the analysis of the problem and the design of the algorithm, making the question of what notation the algorithm is finally expressed in relatively unimportant.† This point is not always valid, we suggest,

* Occasionally there are signs that pedagogical and moralistic considerations enter into the calculists' thinking, as when H. Zemanek says "we do not want the easy application that a natural language offers because the user would then not reflect enough on what he instructs the machine to do."¹ There is a hint here of satisfaction at the discipline imposed by the computer on its users; some suggestion that easy, natural-language access to it would be a tragic waste of the opportunity it offers to straighten out the confused, woolly thinking of nonmathematicians. In other contexts, of course, the boast of the mathematician is that his notation spares him from thinking about what he is doing, allowing him to develop his argument by purely formal manipulation, to avoid the trap of assigning a meaning—a physical interpretation—to intermediate results, and to venture into realms of abstraction where "meaning" in this sense has no meaning.

† See, for example, the discussion appended to Ref. 2.

even for professional programmers—in many routine applications, the filling up of the coding form may take more time and effort than the design of the algorithm, which may in fact be done in the programmer's mind as fast as he codes. But even if the critics' point were always valid, it is irrelevant to the problem for which natural-language programming gives promise of being a remedy. The problem we have in mind is that very many computer customers are perfectly capable of performing the hard part of programming—the analysis of their problem and the design of an algorithm to deal with it—but are frustrated as potential programmers by their unfamiliarity with the easy part—the exact form of the currently acceptable programming languages. We can agree with the critics that this is for professional programmers often the trivial part of their total task; all the more intolerable, then, that it should be for so many thousands of highly-trained (but non-programming) professionals a practically insurmountable barrier between themselves and the machine. These people find it as a rule impossible to practice their own demanding disciplines and also master the myriad details of the programming systems nominally available to them; as a result, they are dependent on professional programmers. These latter are growing ever scarcer relative to both available computing power and the number of potential customers for it. The problem of the programming-language barrier, then, may be a trivial one to the professional programmer, but its practical importance to many others would be hard to exaggerate.

ENGLISH, ACTIVE AND PASSIVE

"Natural-language" programming, it cannot be too strongly emphasized, means programming in whatever terminology is standard for the application in question; this is not necessarily the ordinary English vocabulary. (As we noted earlier, the natural language for numerical computation is algebraic notation.) These pages are largely devoted to that particular natural-programming language that is ordinary English because doing so puts the issues involved in the sharpest relief.

One reason why it can be dangerously misleading to talk about "English-language programming" is that the speaker usually has in mind *active* English, while the listener understands him to mean *passive* English. The difference between them is critical, and confusion on this point is fatal to any possibility of

understanding. Passive English is a language that *reads* sufficiently like English so that almost anyone with some idea of the application involved can understand a procedure described in it, but only those trained in its use (and, usually, with ready access to a manual) can *write* it. The COBOL language is an example of passive English. Its "natural" appearance to the reader of a listing gives little hint of the difficulties of writing it, and the misguided reader who supposed from that appearance that he could use the language without training and reference to a manual would soon discover that it had all the trickiness of any programmers' language.

An active English programming language would permit the free use of a subset of natural English for the *writing* of programs, the subset being open-ended and limited only by those constraints inherent in the application, not by any arbitrary decisions of the system designer's. This means that users would be offered not a fixed number of stereotyped statements to be used exactly as given in the manual except for the replacement of dummy operand names by actual ones, but a stock of words that may be used in any reasonable, straightforward way, and altered or expanded as necessary. No example of this facility can be cited among operational programming systems.*

The distinction is a critical one because the two kinds of English, similar though their appearance on a listing may be, are as far apart in suitability for the role of programming language as can be imagined. For passive English there is simply nothing good to be said. Combining the wordiness and noisiness of a natural language with the rigidity and arbitrariness typical of programming languages, it exhibits the worst features of both and the virtues of neither. (The readability of a COBOL listing is valuable, but has nothing to do with the use of English as a source language; a processor can easily be made to produce such documentation regardless of the input notation.)

English can be justified as a programming language only if it is *active*. (We shall mean active English when we use the term "English" in what follows.) When it can be used actively, its potential importance is immense: it could make programming just enough easier to let many who are now dependent on the services of a professional programmer get

* The writer and a group of associates are engaged in an effort to realize such a system,³ and a few projects reported on in the literature seem to share our approach, in spirit at least.^{4, 5}

at the machine directly. As we noted earlier, the growth of the computer's availability to greater numbers of users is now and for the foreseeable future communication-limited, and every promising attack on this constraint should be energetically pressed. The number of people able and willing to make programming their life's work is probably already near its limit; we can no more expect a professional programmer to be available to everyone who needs the computer than we can provide a chauffeur to everyone who needs a car. And even if programmers were plentiful, there will always be many for whom no amount of professional help would be as satisfactory as direct contact.†

TRANSLATION vs COMPREHENSION

The other great distinction that must be made, just as important as the one between active and passive language, is that between *translation* and *comprehension* of natural language by computer. Many of the skeptics about natural-language programming are mistakenly supposing that the processing difficulties it would involve are as formidable as those that have made fully automatic translation of (for example) Russian to English so elusive. But there is a radical difference between translation—the conversion of one natural language to another—and comprehension—the execution of a procedure described in a natural language. Translation systems treat natural-language statements as data; comprehension systems treat them as commands. In the former the user talks to another human *through* the computer, which is programmed to perform on Russian-language input a transformation under which all information-bearing features of the text are preserved, only their representation undergoing change. The difficulties in doing this are too notorious to need any discussion here.⁶

In a comprehension system the user talks *to* a computer in order to generate coding or to parame-

† The suggestion has been made that the programming problem might be sidestepped by systems that would take the initiative in the man-machine dialogue, presenting to the user at each step a number of alternatives from which he would select one. This technique has its uses, but these seem to be limited to cases where the number of logical alternatives is nowhere greater than about six. Its area of practical application, therefore, is not in the programming of new procedures, but the parameterization of existing, general procedures such as the report-program generator. For broader applications, the CRT screen would have to show the equivalent of a conventional programming manual at each step.

terize existing coding; the required processing of source statements, as compared with that demanded by translation, is simple and straightforward. There are two saving graces in comprehension: one is that the user painlessly forgoes practically all the natural-language features that offer great difficulty. Users of such systems do not care to chat idly with the machine; they want to give data and order that certain procedures be performed on that data. (The procedure specifications may be couched as questions or requests, but are commands nonetheless.) The syntactic complexity of the statements they will want to make is, accordingly, unlikely to be great. Similar considerations suggest that the vocabulary to be dealt with should be within the powers of a realizable system. Users will address such a system within the framework of some particular application, and this is a powerful force for elimination of ambiguity, since it is not often that terms are ambiguous within the vocabulary of a single discipline.

The other saving grace is that the general form of the target language—machine language—is a fixed one, independent of that of source-language statements, and known in advance to the processor. The comprehension process is therefore a restricted many-to-one transformation, not an indefinitely-many-to-indefinitely-many mapping such as translation. The man-machine communication channel offered the user by a comprehension system may be thought of as an ear trumpet worn by the computer, with a wide but well-defined mouth toward the user, and a rather fine-meshed filter somewhere downstream of that mouth.

The practical promise of such a system lies in the indications that the necessary constraints, while powerful enough to spare the comprehension system the most formidable of the problems facing a translation system,* are in no way arbitrary; they should leave the user feeling unimpeded, since they prevent him from doing only what he is little tempted to do in any case. The grotesque sentences that are regularly exhibited in papers on mechanical translation to show how profoundly ambiguous English can be, featuring labyrinthine syntax and words with five meanings (each of which suggests an entirely

* Substantially the same distinction is made by D. M. MacKay.⁷ He is chiefly interested in making clear the inadequacy of what is here called comprehension for completely general communication with the computer, but agrees with us that the "nonlinguistic" treatment of linguistic tokens we propose is sufficient "to enable the computer to accept data and answer questions in verbal form."

different interpretation of the whole)—these have no bearing on the present problem. None of these features is at all likely to appear in a statement composed by a serious worker trying to enlist the computer's help in solving a problem; if once in a great while they do, the system will be doing him a favor by rejecting it.

Further, such a system offers the possibility of a programming language that includes all the semantic and syntactic resources that the users of an application-oriented procedure would naturally employ in invoking it. The building of such a language would be largely done by users themselves, the processor being designed to facilitate the admission of new functions and notation at any time. The user of such a system would begin by studying not a manual of a programming language, but a comparatively few pages outlining what the computer must be told about the location and format of data, the options it offers in output media and format, the functions already available in the system, and the way in which further functions and notation may be introduced. He would then describe the procedure he desires in terms natural to himself.

The system's ability to immediately comprehend (i.e., compile correctly from) his description could be guaranteed only if he restricted himself to those constructions and that vocabulary already introduced to it, and while these conditions will usually be met without conscious effort by any user who has had a few earlier contacts with the system, this high probability of immediate success is not what the system finally depends on. Its real strength is its permanent readiness to be introduced to new notation, even by ordinary users who are by no means professional programmers. The substitution of linguistic open-endedness and cumulative improvability for a fixed notation (whether passive English or a formal calculus) should go far to relieve many users of daily dependence on professional programmers, and in doing so generate a new wave of applications.

OBJECTIONS AND COUNTERPROPOSALS

With this understanding of what "natural-language programming" means, we are prepared to examine the controversy over it. The richest source in print of arguments against it seems to be the writings of Professor E. W. Dijkstra (of the Mathematics Centre, Amsterdam), who has expressed his opposition vigorously, lucidly, and at substantial

length. We will draw heavily for diabolical advocacy on two publications of his;^{8,9} the first of these is a letter that seizes the occasion of the appearance of a report on the MIRFAC¹⁰ system to attack the entire natural-language concept.

The heart of the case Dijkstra makes against natural-language systems is that there is a sharp and ineradicable contrast between human and mechanical response to instructions:

If we instruct an "intelligent" person to do something for us, we can permit ourselves all kinds of sloppiness, inaccuracy, incompleteness, contradiction, etc., appealing to his understanding and common sense. He is not expected to perform literally the nonsense he is ordered to do; he is expected to do what we intended to order him to do. A human servant is therefore useful by virtue of his "disobedience." This may be of some convenience for the master who dislikes to express himself clearly; the price paid is the nonnegligible risk that the servant performs, on his own account, something completely unintended.

If, however, we instruct a machine to do something we should be aware of the fact that for the first time in the history of mankind, we have a servant at our disposal who really does what he has been told to do. In man-computer communication there is not only a need to be unusually precise and unambiguous, there is—at last—also a point in being so, at least if we wish to obtain the full benefits of the powerful obedient mechanical servant. Efforts aimed to conceal this new need for preciseness—for the supposed benefit of the user—will in fact be harmful; at the same time they will conceal the equally new possibilities in automatic computing of having intricate processes under complete control.

The picture Dijkstra draws of man addressing machine is in fact quite unrealistic, and the oversight he commits explains much of the misunderstanding between him and those whose views he is attacking. (For that matter, the latitude that he thinks permissible in giving orders to humans would be astounding to an army officer, doctor, or anyone with experience in directing men in the performance of complex procedures.)

The nature of machines and their response to instruction is not properly at issue, because programmers practically never address machines directly; they address programming systems, which act as buffers between them and the less amiable characteristics of computers. These systems already permit many users to disregard for almost all purposes the particular model of computer on which their programs are to run, and there is every reason to think them capable of much further development in the direction of liberating the programmer from concern

with the purely mechanical, should we decide to. Any appeal to the characteristics of naked machinery, then, begs the question; the *effective* machine, the machine as it is known to almost all programmers, is very nearly whatever we want it to be—and what we should want is a philosophical, not a technical, question.

REDUNDANCY

One of the ways in which the bare computer's demand for precision and explicitness can be mollified is through one of the very characteristics of natural language that calculists most abhor: redundancy. In many other technologies, the word "redundancy" is an honorable one, standing for a means to reliability. Electronic circuits, mechanical structures, communication channels, and hydraulic lines are made doubly and triply redundant, and users are rewarded by diminishing failure rates; why should this technique have been so unsuccessful when applied to programming? The answer, of course, is that it has not been unsuccessful, merely untried. We have long been playing a game that might be called "But Don't Tell the Computer!"; the point of which apparently is to see how little information we can give the computer and still get some output. If we tell the same thing twice to one of the programming systems designed to play this game, or tell it something it does not demand to be told, it not only fails to avail itself of the possibly vital information being offered, but may well gag, stop compiling, and disgorge a core dump at us. What is loosely called redundancy in programs is taken simply as noise by today's systems, when they accept it at all; no system in common use today is capable of profiting from real redundancy. Programming systems that know how to use redundancy would reward programmers with the same resistance to minor failure other redundancy-exploiting systems exhibit, using the extra information offered it at one place to compensate for minor omissions and inconsistencies elsewhere in the program. As a trivial example for those to whom this notion is utterly foreign, consider a program unit that contains two statements implying that the value of X is A, and one that implies its value is B; the conventional system either fails to notice the contradiction and proceeds to compile a faulty program, or notices it and merely throws the problem back at the programmer. A system capable of using redundancy

would use majority-decision logic to conclude that X's value should probably be A (and would, of course, notify the programmer of its assumption).

Dijkstra's attitude toward redundancy is a mixed, even a troubled one. He is here as elsewhere more thoughtful than most of the calculus school in seeing some positive value in it, but differs sharply from our position because he judges it solely as a device for optimizing the object program's use of space or time, rather than as a means of improving man-machine communication. Judged by the criterion he employs here, redundancy is clearly a tricky thing, perhaps better avoided; if it can sometimes improve the object program, it also complicates the processor and consumes translation time. He accordingly concedes it some value, but warns:

. . . if the redundant information is to be a vital part of the language, the defining machine *must* take note of it, i.e., it must detect whether the rest of the program is in accordance with it and this makes the defining machine considerably more complicated.

and later adds:

But we can hardly speak of "good use of a computer" when the translator spends a considerable amount of time and trouble in trying to come to discoveries that the programmer could have told it as well!

His final position is that redundancy is permissible, but is to be kept optional; optional not only in that programmers need not use it, but in the deeper sense that processors need not use it if offered, and should not require it to produce good object programs.

Given his premises, Dijkstra's conclusions are unavoidable—but those premises are questionable, and Dijkstra himself, as will be shown, questions them by implication elsewhere in the same paper. The rule suggested by the second of the above-quoted remarks, that the processor should be spared anything the programmer can do for it, is absurd if taken literally; unless checked by some superior principle, it cuts the ground from under all software. But Dijkstra does not, of course, intend this; he has on an earlier page rebuked those who by "good use of a machine" mean simply use that is economical of time and space, saying:

I have a suspicion, however, that in forming their judgment they restrict themselves to these two criteria, not because they are so much more important than other possible criteria, but because they are so much easier to apply on account of their quantitative nature. . . . there is sufficient

reason to call for some attention to the more imponderable aspects of the quality of a program or of a programming system.

He concludes this section of his paper by saying, in words with which the present writer is thoroughly in accord: "in the last instance, a machine serves one of its highest purposes when its activities significantly contribute to our comfort."

This is the superior principle that provides justification for the use to which we would put redundancy, as it does for the existence of software in general.

WHAT CALCULUS, AND WHY?

Dijkstra aligns himself with Professor John McCarthy in calling COBOL "a step up a blind alley on account of its orientation towards English which is not well suited to the formal description of procedures." It should be noted in passing that we are in agreement with McCarthy and Dijkstra in thinking the COBOL language unsatisfactory, but for reasons diametrically opposed to theirs: what we see as wrong with it is precisely its lack of "orientation toward English"; the COBOL language, as a passive subset of English, is as unacceptable from our point of view as from theirs. Dijkstra says of such languages that "giving a plausible semantic interpretation to a text which one assumes to be correct and meaningful, is one thing; writing down such a text in accordance with all the syntactical rules and expressing exactly what one wishes to say, may be quite a different matter!"¹¹ His diagnosis of the trouble with passive systems is astute, but the cure (insofar as language can offer one) is to make the program easier to write, not harder to read.

The calculus school would undoubtedly regard active English as even less suitable for the "formal description of procedures" than the passive type; what they strikingly fail to say is what language *would* be suitable for that purpose, and what programmers would be capable of using it. It seems clear that whatever the calculus school has in mind for programmers, it is something far more rigorous and succinct than they now use, something much closer in spirit to mathematical notation—McCarthy's use and Dijkstra's citation of the ominous phrase "formal description" is probably sufficient indication. The belief of the calculists that mathematical notation constitutes a distinct language that makes substantive error harder to commit (or

easier to find) is simply unfounded, however. The calculists have overlooked or forgotten the ancestry of their notation: both historically and logically, mathematical notation is an encoding of a subset of natural language, and is not an independent language.* The elementary operations and operands of mathematics can only be defined in one of the natural languages; more elaborate operations and operands are defined in terms of the elementary ones (and, where necessary, further natural-language definitions). If English is incorrigibly imprecise, then all mathematical notation is congenitally infected with that imprecision, for English is the ground from which it sprang and to which it still returns at frequent intervals for support.¹²

Mathematical notation is, in fact, nothing more than a shorthand† that facilitates the very compact graphic expression of natural-language statements belonging to a certain special universe of discourse.

* A *language* is the direct symbolic representation of an original and independent anatomization of experience into objects of interest. As such, it almost certainly will not map perfectly into any other language—as French, for example, cannot be mapped element-for-element into English. A *code* is a derivative representation, artificially created through the application of some transformation to a language or subset of one. This derived representation has a completely determinate relationship to the language from which it derives, and neither adds to nor subtracts from it any information. Its practical advantages may be great, but whether these lie in economy, security, or convenience, they do not spring from the code's supposed superiority as a representation of reality, but from its intrinsic properties, such as brevity, secrecy, or mnemonic power.

† Many mathematicians claim for their notation a considerable heuristic power, crediting it with suggesting to them relationships and developments they might otherwise have missed. Their notation clearly has such power for practitioners steeped in its use, but it is hardly unique or even highly unusual in this. Poets, for example, have testified that they often find a line or word they have just written suggesting a successor that would not have occurred to them but for some feature of their "notation"—rhyme, alliteration, pun, etc. The point is that any notation, indeed any tool, whose user is saturated in it comes to be so much a part of his nature that it seems sometimes to work of its own accord. While there remain better and poorer notations, then, the distinction between them does not hinge on this common property of seeming to come alive in an experienced user's hand.

Not only is it useless as a differentiator between good and bad notations, but its value as a means of arriving at significant results in mathematics has been questioned at the highest level. Gauss, for example, had occasion to reproach his contemporary, Waring, for inordinate emphasis on the heuristic value of notation: Waring had said of certain theorems that they were very hard to prove because of the "absence of a notation to express prime numbers." Gauss, himself the inventor of the standard sign for the congruence relationship, replied sharply that mathematical proofs depend on *notions*, not *notations*.¹³

In principle, all mathematical discourse could just as well be carried on in the ordinary vocabulary of which that notation is simply a condensed representation. In practice, the notation is indispensable because it permits the expression in graspable form of propositions that, expressed verbally, would be too tenuous, too rarefied and linear to be apprehended as coherent wholes.

But such propositions so expressed do not in practice form the fabric of mathematical arguments. The notation as it is actually used in mathematical papers is more a medium for the presentation of intermediate results, while the burden of exposition is invariably carried by a prose narrative typically starting "Let L_1 be a . . ." and ending ". . . which completes the proof." The equations and other symbolic expressions embedded in the prose serve to record and isolate for possible inspection the important milestones along the way; their study is seldom necessary for comprehension of the argument, and unless the reader suspects an error in formal manipulation he will not dwell on them. Supporting this observation is the fact that a non-Russian-speaking mathematician will generally be unable to read a paper in a Russian mathematical journal. That mathematicians need to learn foreign languages or employ translation services just as much as any other scientists shows that little of the meaning of even the most formal mathematical discourse is carried by the internationally accepted notation of the discipline.

Insofar as the call for a programming calculus is a demand that any programming insights we manage to achieve should be, as far as possible, incorporated into our programming tools, it is unexceptionable. But we must beware of assuming that mathematical notation is analogous to programming notation when it is only homologous to it; if we are to compare things of like function rather than things merely of like form, the parallel between them breaks down, since they play different roles in their respective worlds. Any analogy between programming and mathematics then, is hazardous (to the extent that mathematical notation has an analog in programming, it would be the "comment" facility offered by most programming systems).

Consider, for example, just these two differences:

1. Mathematical objects are imaginary and arbitrary; programming objects are, in general, real and given. A mathematical object is created by fiat,

and has exactly and only those qualities that are given it explicitly (and those logically implied by them). The objects of ultimate interest in programming, other than the special subset that are also mathematical, preexist in nature, and can neither be altered nor exhaustively described.

2. Mathematical goals are flexible and opportunistic; programming goals are fixed and predetermined. If a mathematician trying to prove an intuitively obvious theorem finds it not merely impossible to prove true, but in fact demonstrably untrue, he is far from disappointed; he has a much more important result than he was trying for. A programmer unable to reach his original goal is simply a programmer defeated.

These differences are so fundamental and far-reaching that any argument founded on the resemblance between a proof and a program should be held suspect; it is particularly likely to be specious if it assumes that that resemblance implies an identity of goals, methods, or problems in the procedures that underlie the two.

These remarks, it should be needless to say, are not offered as a definitive treatment of the many problems—linguistic, psychological, metamathematical—that they touch upon. They are intended only to suggest that there are more and deeper issues involved in the notational question than are covered in the usual easy antithesis between natural sloppiness and formal precision, and that it is far from clear that a formalism patterned on mathematical notation is the answer to any burning problem in practical programming.

CONSEQUENCES FOR DEBUGGING

Each of the two schools of thought claims that the adoption of its proposed type of language would make for a significant advance in debugging, whether by making errors harder to commit or easier to find. Gawlik, in the paper that incited Dijkstra to the writing of the letter we have been quoting from, claimed for MIRFAC that programs written in it could be checked for correctness by anyone who understood the problem, even if he knew nothing of programming:

MIRFAC has been developed to satisfy the basic criterion that its problem statements should be intelligible to nonprogrammers, with the double aim that the user should not be required to learn any language that he does not already know and that the problem statement can be checked for correctness by somebody who understands the problem but who may know nothing of programming.¹⁰

Dijkstra, as we have seen (in the preceding section), rejects this contention, but says in outlining his own ideas on language design:

In particular I would require of a programming language that it should facilitate the work of the programmer as much as possible, especially in the most difficult aspects of his task, such as creating confidence in the correctness of his program.¹⁴

Both parties, then, beneath their conflict over notation, agree that a good language would go far in helping programmers with debugging; both, we think, are wrong.

There are two kinds of error one can make in writing a program: the formal and the substantive. The formal errors result from infractions of rules for using the language; in another familiar nomenclature, they are known as syntactic errors. Substantive errors are those that prevent the procedure embodied in the program from solving the problem—either because it does not really say what the programmer intended, or because what the programmer intended is wrong. It is clear that the debugging advantages claimed for their language types by Gawlik and Dijkstra alike must be limited practically exclusively to the formal errors. Neither claims that a language of the type he proposes would make it impossible to mistake the problem or misstate its solution. Even the weaker claim that they would make the detection of these substantive errors significantly easier cannot be allowed; it is a common experience among programmers to desk-check their programs for hours, only to find that they have repeatedly passed as correct an error they will later call “obvious,” recommitting at each iteration the mental lapse that gave rise to the error originally. The suggestion that the programmer should have a colleague check his program is unrealistic; it amounts to nothing less than a demand that each problem be programmed twice, since an independent checker is useful only if he is, at least mentally, programming the problem in parallel as he reads the original.

But the formal or syntactic errors which both parties in this controversy promise to eliminate or minimize are by far the less important class of errors in a compiler-language program, and many processors

already offer detection of all such errors in the first compilation of the program, so even if the promised advantage should materialize, it will not be worth giving up much to get. Dijkstra's own words support our position. In a later passage from the paper cited above, he recognizes the two kinds of error described here, and grants that only a response from the party addressed (human or mechanical) can reveal substantive error, but he then unaccountably dismisses this genus from consideration as if it were either negligible or irremediable:

... we badly need in speaking the feed back, known as "conversation." (Testing a program is in a certain sense conversation with a machine, but for other purposes. We have to test our programs in order to guard ourselves against mistakes, which is something else than imperfect knowledge of the machine. If a program error shows up, one has learnt nothing new about the machine—as in real conversation—one just says to oneself, "Stupid!")¹⁵

The self-directed cry of "Stupid!" is a familiar one to programmers; it is traditionally uttered after long searching of dumps and listings for a clue as to why a syntactically perfect program ran wild or gave wrong answers. And it is in finding these substantive errors, and not those that any compiler will pinpoint on the first run, that the programmer desperately needs help. Since no one has shown how a wise choice of source language offers any help to speak of in dealing with substantive error, we conclude that debugging is not an important consideration in the design of a programming language, or in choosing one from among others on the same logical level.

This is not to say that nothing can be done to help programmers with their debugging problem; we have elsewhere described what a properly designed programming *system* can do in that regard.¹⁶ For the problem of ensuring that the procedure to be executed is formally correct, without loose ends or inconsistencies, the most attractive solution so far is the rigorous tabulation of the decision rules implicit in the procedure. With such a tabulation, the detection of this important subclass of errors can be reduced to the mechanical checking of a table for completeness and consistency, and such a representation of the procedure can even be used directly as computer input.¹⁷ (Whether it is desirable to use it so is doubtful, since the tabular arrangement throws a strong light on formal error at the price of obscuring the practical meaning of the program, which is better conveyed by a narrative form. It may

well turn out that such decision-rule matrices, like COBOL-language statements, are not really wanted as a source language, but as a by-product of compilation—in short, as output rather than input.)

SPECIMEN CONFUSIONS

Much of what little literature exists on the topic of natural-language programming, whether pro or con, suffers from failure to take account of elementary distinctions of the kind insisted on here. Two examples, chosen not for their egregiousness but merely for simplicity and brevity, will suffice as illustration. Their author* makes the familiar contrast between sloppy English and a precise calculus, then gives what he takes to be supporting evidence:

The written and spoken English of the average adult is imprecise, often redundant and incomplete. Compared to mathematics and symbolic logic it is a poor vehicle for the expression of thoughts in precise, logical form. In one experiment, for example, the subjects were shown the single logical premise: "What can you say about B if you know that all A are B?" The subjects tended to continue by concluding that all B are A. In other studies it has been demonstrated that verbal habits operate as a substitute for thought and often lead to errors in logic. Evidence of this type might lead one to reject a Near-English language as a medium for man-computer problem-solving.¹⁸

There are several kinds of error entwined in this brief passage. The experiment, if correctly reported, was ill-designed. We are asked to accept its results as showing that symbolic-logical notation is better than English for conveying precise notions; what it does show is merely that some people do not understand the idea of set membership. Evidence that any greater number of them would have understood the relation between A and B if it has been expressed as " $A \subset B$ " is not given; common sense suggests that no such evidence exists. Certainly all who recognize the expression " $A \subset B$ " would know that the relation is not symmetrical—but just as certainly all of them, plus some who are not familiar with such notation, would understand this from the English version. So if the experiment had been run properly, with a control group, more of those given the problem in English would have gotten it right than those given it in technical notation. More striking yet, it is far from clear how the rather complex question that Van Cott calls a "single logical prem-

* Dr. H. P. Van Cott, Associate Director, Institute for Research in Organizational Behavior, Washington, D. C.

ise" could be stated in technical notation; what is the symbol for "What can you say about . . .?" It would seem that not only might fewer people have given the right answer if English had been ruled out, but that the problem could not even have been put to them without its use. Insofar as this anecdote suggests anything, it would tend to show that English has some powers that even a symbolic logician might be unwilling to forgo.

Van Cott offers further evidence against English:

The rationale for the development of Near-English user languages was based on the assumption that the human could not adapt to a new, more formal language easily, rapidly or with any degree of reliability.

Anecdotal evidence suggests that this assumption may be false. For example, people rapidly adapted to the use of the telegraph as a means of communication—reducing the redundancy and ambiguity characteristics of normal English in order to reduce message lengths, save money and avoid misunderstanding.¹⁹

It is not clear who the "people" are who adapted to the telegraph: are they the professional telegraphers who pounded the key all day long, or the customers who passed their hand-printed messages over the counter to be transmitted? It is just possible that Van Cott means the former, since it does not appear that the public had to do any adapting at all (unless to improve their penmanship), but it hardly seems warranted to make any broad statement about "people" if it is only the comparative handful of highly practiced operators who are the sample group. If he is referring to the telegraphers, we have here another example of the confusion of *language* and *code* that we pointed out in our discussion of mathematical notation. All a telegrapher does is to encode a natural-language message, not translate it into another language; the language remains English, but in a different physical representation, and no conclusions about human adaptability to a "new, more formal language" are warranted.

But it seems far more likely that Van Cott is referring not to the telegraphers, but to the users and their resort to "telegraphese," meaning "language characterized by terseness and elliptical expressions such as are common in telegrams." If so, one set of objections is replaced by another, yet more devastating. Before drawing Van Cott's or any conclusion from this observation, the assertion that "people rapidly adapted" to this new language must be examined. Of course customers paying by the word often tried to minimize the number of words in their

messages, even at the price of taking more care in their composition. The disproof of their adaptation, however, is given by their immediate reversion to customary habits of speech and writing as soon as this pressure was removed. If this "new language" had any merits other than that of saving money under the rate structure fixed by the telegraph companies, the general public evidently failed to see them. But let us waive this objection; the most curious thing about this second count in Van Cott's indictment of English is that, like his first, it demonstrates the value of English if it demonstrates anything. If telegraphese is an example of "a new, more formal language," then we have some unlooked-for evidence that a subset of English makes a good vehicle for the economical conveyance of clear ideas.

CONCLUSION AND SUMMARY

There has been no attempt in this paper to make a complete case for natural-language programming, but only to clear away the greatest of the misconceptions and confusions that have long impeded useful discussion of the subject. The writer's original intention of disposing of these incidentally in the course of illustrating the positive advantages of natural language had to be dropped as it became apparent how many and deep-seated these misconceptions were, and how much analysis was needed to show them as such. The more positive side of the argument has had to be deferred, but should be published at an early date.

Chief among the points we sought to make here are:

1. Natural-language programming is an attempt to put nonprogrammers in direct touch with the computer, not to spare the advanced professional programmer what may be to him an insignificant part of his total job.
2. A natural programming language is one that can be written freely, not just read freely.
3. The task to be performed by the processor for such a language is qualitatively different from that of translating one natural language to another.
4. The redundancy of natural language is one of its greatest potential advantages, not a prohibitive drawback.

5. Finally, the possibility of user-guided natural-language programming offers a promise of bridging the man-machine communication gap that is today's greatest obstacle to wider enjoyment of the services of the computer.

ACKNOWLEDGMENTS

Valuable criticism of early drafts of this paper came from Daniel L. Drew and Allen Reiter of Lockheed Missiles & Space Company, Edward Theil of the Department of Mathematics, University of California at Davis, and Christopher Shaw of System Development Corporation. Although these friendly critics have much improved the presentation of our argument, it must not be assumed that any of them subscribe to it.

REFERENCES

1. H. Zemanek, "Semiotics and Programming Languages," *C. ACM*, vol. 9, no. 3, pp. 139-43 (Mar. 1966).
2. J. E. Sammet, "The Use of English as a Programming Language," *ibid*, pp. 228-30.
3. M. I. Halpern, "A Manual of the XPOP Programming System," Lockheed Missiles & Space Co., Palo Alto Research Laboratory, Oct. 1965.
4. J. Weizenbaum, "ELIZA—A Computer Program for the Study of Natural Language Communication Between Man and Machine," *C. ACM*, vol. 9, no. 1, pp. 36-45 (Jan. 1966).
5. A. P. Yershóv, "One View of Man-Machine Interaction," *J. ACM*, vol. 12, no. 3, pp. 315-25 (July 1965).
6. Y. Bar-Hillel, *Language and Information*, Addison-Wesley, Reading, Mass. 1964, pp. 153-84.
7. D. M. MacKay, "Linguistic and Non-Linguistic 'Understanding' of Linguistic Tokens," RAND Corporation, Memorandum RM-3892-PR (Mar. 1964).
8. E. W. Dijkstra, "Some Comments on the Aims of MIRFAC," *C. ACM*, vol. 7, no. 3, p. 190 (Mar. 1964).
9. —, "On the Design of Machine Independent Programming Languages," in *Annual Review in Automatic Programming*, (R. Goodman ed.), Pergamon Press, New York, 1963, Vol. III, pp. 27-42.
10. H. J. Gawlik, "MIRFAC: A Compiler Based on Standard Mathematical Notation and Plain English," *C. ACM*, vol. 6, no. 9, pp. 545-48 (Sept. 1963).
11. Ref. 9, p. 31.
12. Florian Cajori, *A History of Mathematical Notations*, 2 vols., Open Court, Chicago, 1928.
13. Constance Reid, *From Zero to Infinity*, 3d ed., Crowell Apollo Editions, New York, 1966, p. 129.
14. Ref. 9, p. 30.
15. *Ibid*, p. 33.
16. M. I. Halpern, "Computer Programming: The Debugging Epoch Opens," *Computers and Automation*, Nov. 1965, pp. 28-31.
17. [Wim Boerdam et al], "DETAB/65 Language," Working Group 2, SIGPLAN, Los Angeles Chapter, ACM (June 1964).
18. H. P. Van Cott, "Flexible Machine Language for Commander-Computer Chats May be Key to Flexible C&C," *Armed Forces Management*, vol. 11, p. 95 (July 1965).
19. *Ibid*, p. 95.

EXPLICIT PARALLEL PROCESSING DESCRIPTION AND CONTROL IN PROGRAMS FOR MULTI- AND UNI-PROCESSOR COMPUTERS

J. A. Gosden

*AUERBACH Corporation
Philadelphia, Pennsylvania*

This paper discusses the development and current state of the art in parallel processing. The advantages and disadvantages of the various approaches are described from a pragmatic viewpoint. One that is described and discussed is a natural extension of the evolutionary approaches. This alternative covers the problems of locating, describing, controlling, and scheduling parallel processing. It is a general-purpose approach that is applicable to all types of data processing and computer configuration.

GENERAL APPROACH

The approach is based on the following assumptions.

1. The specification of potential parallel activity in processing will depend largely upon programmer specification.
2. Major benefits will accrue only if a high degree of parallelism exists; i.e., either many parallel paths exist or, if the parallel paths are few, they must be long.
3. Programmers will specify parallelism only if it is easy and straightforward to do so.

4. A simple control scheme is needed to provide straightforward scheduling.

Programmer Specification

It is obvious that an automatic scheme can be devised to recognize independent parallel paths when the paths are at a task or an instruction level. The latter scheme has been implemented in STRETCH^{9*} and other computers; the former is a simple matter of comparing the names of the input and output files of various tasks in a job.

At intermediate levels where subroutines, parameters, and many other forms of indirect addressing exist, the problem is at present unsolved from a general-purpose practical viewpoint.

Therefore, we must now look to the programmer to define independent processes.

Degree of Parallelism

Previous experience with simultaneous or overlapped I/O and multiprogramming suggest that full potential gains are never realized because of the

* In this paper the references are combined with an extensive bibliography and are, therefore, listed in their alphabetical sequence, rather than in consecutive numerical order.

difficult scheduling problems. Therefore, we expect that scattered parallelism of two or three short paths may not lead to significant gains; multiple paths must be sought. If there are two or three long parallel paths, it may be more practical to specify them as separate tasks.

Easy Specification

Experience indicates that any new feature in parallel processing must be easy to specify; otherwise, the feature will not be much used or exploited. Both simultaneous I/O and multiprogramming gained most when they were made programmer independent. However, our first assumption implies that in this case we cannot rely on programmer independence, so we must make the specification of parallel processing easy and simple to encourage its use.

Good Control Scheme

There are two main requisites of a good control scheme. First, it must be simple so that the overheads of implementing it do not seriously erode the advantages gained. Second, it should be efficient with simple ad hoc scheduling algorithms. Experience has shown that ad hoc schemes are successful whereas complex schemes are not only difficult to implement but disappointing in general performance.^{9,13}

BACKGROUND

This section discusses some of the existing literature on parallel processing and relates the findings to the assumptions previously made.

Parallel processing in various forms has produced interesting developments in the computer field. For those interested in pursuing the subject, a general bibliography attached to this paper indicates the continuing voluminous literature on this general subject and its pervasive influence on computer hardware and computer software architecture. It is possible to divide the development into six general classes:

- Class 1 (1953) Simultaneous Input-Output
- Class 2 (1957) Fork and Join Statements
- Class 3 (1958) Multiprogramming Operating System
- Class 4 (1959) Processor Arrays
- Class 5 (1961) Planned Scheduling
- Class 6 (1962) Re-entrant On-demand System

The dates are approximate origins of development, but there has been no attempt to assign single inventors because most ideas were developed in several places more or less concomitantly. These titles are useful labels for discussion in this paper but are not intended as formal definitions. The author regrets that he has not been able to incorporate in this paper formal definitions for the various labels used.

Class 1, Simultaneous Input-Output (achieved either by the use of hardware or even programmed time-sharing of the processor) made useful increases of two- or threefold in computer power by parallel use of input-output and computation processes.

Class 2, Fork and Join Statements provided a simple language mechanism for programs to take advantage of a multiprocessor computer system. The literature does not show great gains in this area. Examples given^{12,17} have not looked rewarding except one attributed to Poyen⁴⁸ by Opler⁴⁵ which, however, is somewhat clumsy; it is predicated on a specific number of processors and requires an exertion on the part of the programmer that will not in general be obtained. Opler, however, has recognized what we shall call "parallel-for" construction; his term is "do-together."

Class 3, Multiprogramming Operating System provides a mechanism for a greater degree of simultaneous input-output by mixing programs and balancing the use of the parts of a computer configuration,^{14,26,27,56} which has resulted in useful gains of some two- to threefold.⁴² This subset of parallel processing maintains independent processes each of which is part of an independent job. These systems have enjoyed extensive development in both large and small computer systems, and in general their schedulers operate on a queue of tasks using ad hoc scheduling algorithms. Good results have been achieved with straightforward general-purpose approaches.

Class 4, Processor Arrays are computers with replicated arrays of processors which are powerful on a specific range of problems.⁴⁴ These processors operate on an array of data obeying a common program. The SOLOMON⁵⁴ computer and the structure proposed by Holland²⁸ are the best known although they have not enjoyed popularity, partly because they depart radically both in architecture and solution techniques from the evolutionary mainstream (a tendency not likely to succeed as Brooks⁷ pre-

dicts) and partly, in the writer's opinion, because they shun the qualifier "general-purpose," which has been the touchstone of the success of the computer. Associative memories are another special case.

Class 5, Planned Scheduling is a title which covers schemes that propose to describe parallel-processing structures and to schedule them using a CPM or job shop scheduling approach. Typically, each part of a process is described separately and is associated with a set of conditions or inputs which cause it to be executed instead of being linked in a program sequence.^{21,37,41,52} Such schemes are the antithesis of Class 3 in that they pre-partition the problem into logical parts rather than apply the somewhat arbitrary paging of general operating systems. Some of those schemes attempt (or attempted¹³) pre-scheduling, relying on expected time and space utilization estimates. Again this is a major departure from the mainstream of computer development and has not enjoyed much popularity.

Class 6, Re-entrant On-demand Systems are an interesting mutation of multiprogramming systems where the process is common to all users but the data sets may be independent. The classic example is an airlines reservation application. Since the queries are independent, a multiprogramming operating system approach can be used. The principle of re-entrant routines was developed to allow sharing of access to a common set of programs.^{3,5,49}

It is also relevant to point out a class of activity which has not existed, that is, automatic recognition of parallel processes. This is a subject which is still at an early stage of development. At present there are systems that seek out some overlap at the instruction level in large-scale processors, but no work has been done at the subroutine or program level. Even at the instruction level, as the processor reshuffles access and attempts parallelism, as in the look-ahead in STRETCH,⁹ the problem of interrupts and unwinding is formidable. The problems of finding implicit parallelisms in programs are also formidable, especially with the use of parameters, block structures, and late binding. Even if possible, the compiling overhead in analysis may be considerable.

The foregoing summary identifies the author as a mainstream evolutioner rather than a radical innovator. This does not mean that he designates classes 4 and 5 as useless but as yet unproved, and agrees

that radical innovation must prove itself by large gains to offset its disruption whereas evolution can be justified by more modest gains.

THE PARALLEL FOR

The simple premise of this paper is that all FOR statements in an ALGOL program, and similar constructs in other languages, are good potential sources of parallel activity. FOR statements divide into two kinds: first, those which are iterative and second, those which are parallel. The second group is a large fraction of the total. Two examples may suffice, one scientific and one commercial.

A typical scientific example is matrix addition in which the pairwise additions of corresponding elements can be performed independently, and therefore in parallel. A typical commercial example is the extension of an invoice where the individual multiplications of price per unit times quantity for each item can be performed independently, and therefore in parallel.

It is easy to see that by using the simple device of describing each FOR statement as either iterative or parallel, a large group of parallel paths in programs can be identified. It is interesting to note that Opler,⁴⁵ and later Anderson,² use a FOR statement to illustrate their notations.

The FOR statement is a special case of a loop; it is, therefore, natural to dichotomize loops into those that are either iterative or parallel in nature. One simple example shows that parallel loops not only exist but exist in large numbers. The main loop in a payroll program (that is, the payslip loop) is independent for each man. It is, of course, a FOR loop where the meaning might be "for every record on the master file. . ."

It may appear unreasonable to distinguish between a PARALLEL FOR and a rewritten loop, but there are substantial pragmatic differences that would distinguish them at a programming language level.

SUMMARY OF PARALLEL STRUCTURES

We have identified three areas of parallel structure in programs:

1. FORK to several dissimilar paths and JOIN
2. PARALLEL FOR statements
3. LOOPS rewritten as PARALLEL FOR statements

All of these structures require some kind of JOIN as a prelude to subsequent processing. Techniques for Item 1 have been described elsewhere^{12,17} and are described later for Items 2 and 3. However, there also exist two other types that should be recorded for completeness:

4. FORKS without a JOIN
5. Completely independent processes

As an example of Item 4 consider a task that updates a file and then produces several reports. After the update there is a FORK to each report routine with no need to JOIN. PL/1 has a facility to do this.⁴⁹

As an example of Item 5 consider an airlines reservation system and the individual queries and updates made upon it.

ADVANTAGES OF PARALLELISM

The identification of parallelism in programs is useful only if some advantage is gained thereby. For both uni- and multiprocessor systems there are advantages.

In general we can note that, whereas program structures in which FORK statements are used do not tend to show a high order of parallelism, PARALLEL FORS and PARALLEL LOOPS do naturally show a high degree of parallelism; moreover, because they tend to occur in nested sets, they increase the parallelism multiplicatively.

Multiprocessor Advantages

The multiprocessor situation contains four interesting cases: First, there is the simple case where the number of programs to be run is less than the number of processors. The availability of parallel paths enables the system to use the otherwise idle processors. This situation may not seem to occur frequently but it can occur often when a system begins to run out of peripheral devices, even if the load on them is not heavy.

Second, there is the case where a mixture of high- and low-priority jobs is being run. If the disparity in processing is large enough, several processors could work on the high-priority program and reduce its turnaround time.

The third and fourth are cases where batch processing response times for large jobs can be improved if the jobs are run serially rather than in parallel.

The third case is illustrated by the simple (and over-simplified) example of three jobs that each require one hour's capacity of the system. When run serially, they would be completed in one, two, and three hours, respectively, giving an average turnaround time of two hours. When run in parallel, they would have an average turnaround time of three hours. Of course, the choices are never as simple as this example supposes but, in general, serial operation is desirable.

The fourth case considers the savings in internal system overheads when parallel activity on one task replaces parallel activity on different tasks. In situations where frequent overlays are made for both data and program segments, the amount of shuffling can be reduced. In particular, if several processors keep approximately in step (not necessarily exactly in step as required in SOLOMON), they can share access to, and residence of, program segments and some data segments. Even if they do not share exact access, but are accessing a disc, cartridge, or other complex access device, then there is more opportunity to batch and optimize accesses for parts of one task than with parts of diverse tasks using different areas of auxiliary storage.

Single Processor Advantages

The single processor system contains two interesting cases. First, there is the case of a mix of multiprogrammed tasks where one has an outstanding priority. Suppose it stalls due to some wait for I/O; then if parallel paths exist, it may be able to proceed on another path instead of passing control to a lower priority task. Such cases have been "hand-tailored" in the past relying on a complex look-ahead type of structure or the batching of accesses to auxiliary storage. If parallel paths were started automatically, the batching could adjust itself dynamically.

A typical example would be an operating system that has two kinds of scheduler, one that allocates processors and another that batches and queues access to auxiliary storage. Then parallel paths could utilize processors while more urgent processes are awaiting input.

Second, there is the case of reducing overlays. Particularly in systems where relatively small overlays are used, the problem of fitting internal loops within one overlay is difficult and sometimes impossible. On a loop that needs two segments, the overlays needed for x executions could be reduced from

$2x$ to $2x/n$ if n parallel paths were used. Both these cases also apply in the case of multiprocessor systems, and are illustrated in the example given in the section on Processor Scheduling.

TECHNIQUES

Basic Elements

This section is not a comprehensive treatment of all cases but a range of examples to show how simple techniques can be used to implement and control parallel activity. It will be seen that all the techniques can be handled by a simple organization of five simple functions (PREP, AND, ALSO, JOIN, and IDLE). This provides a uniform mechanism which can easily be incorporated in processor hardware if the economics justify it.

PREP is a function performed before a new set of parallel paths begins. It causes a variable called PPC (parallel path counter) to be established. If other PPC's exist for this process, they are pushed down, which allows sets of parallel paths to be nested. PREP sets the new PPC to the value one. *AND* (L) is a simple two-way fork; it requests the controller to start a parallel sequence at L and to add one to the current PPC for this process. The processor executing the AND continues to the next instruction. The controller queues the request for the AND sequence.

ALSO (L) is a simple two-way fork which is the same as AND except that no PPC converters are involved. It is equivalent to TASK in PL/1⁴⁹ and is used for divergent paths that do not rejoin. *JOIN* is a function used to terminate a set of parallel paths. When executed, it reduces the current PPC for the process by one. If PPC is then zero, it is popped up and processing continues. If PPC is not zero, meaning that more paths remain to be completed, it releases the processor executing the JOIN.

IDLE is a function that ends a parallel path and releases the processor executing the IDLE.

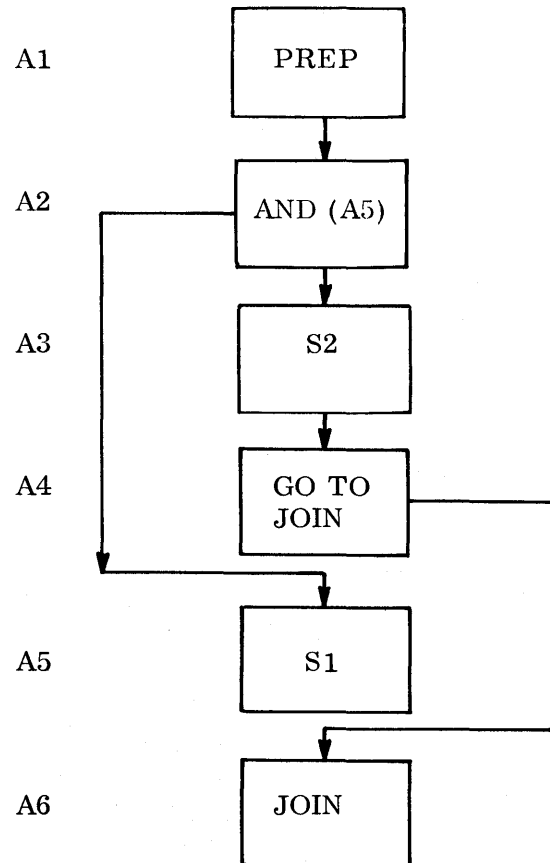
It should be noted that by associating PPC with a process rather than the JOIN statement, the property of re-entrant code is retained. JOIN is used as a device similar to CLOSE PARENTHESES.

Example 1. Classic FORK and JOIN

The popular form of the FORK statement is:

FORK (S1, S2, S3 . . .) (J)

At the execution of this statement the program divides into parallel paths that commence at statements labelled S_i , and join at the statement labelled J .

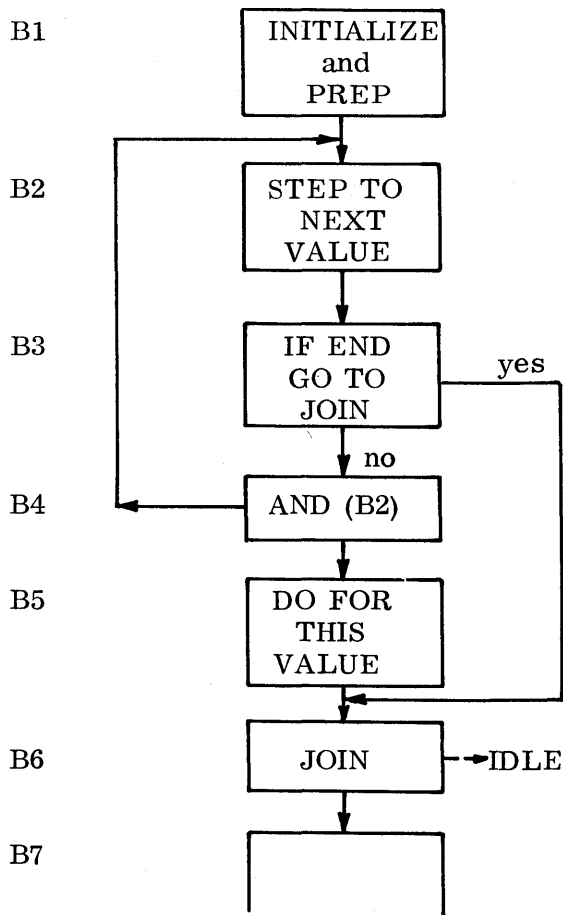


Example 1 shows the implementation of FORK-JOIN for two paths. A1 establishes a new PPC counter and sets the counter to 1. A2 initiates A5 as a parallel path and increments the PPC counter to 2. A3 executes the second path S2. A4 ends S2 with a transfer to JOIN. A5 executes the first path S1. A6 is the join; when one path ends it reduces the PPC to 1 and frees the processor, when the other path ends, the PPC goes to zero, the next PPC is popped up and the processor continues to A7. The extension to three or more paths is obvious.

Example 2 shows the implementation of a PARALLEL FOR process. Boxes B1, B2, B3, and B4 are the translation of the FOR statement. B1 initializes the generation of the values of the parameter of the FOR statement and establishes the new PPC. B2 is the incrementing or stepping function. B3 is the end test and transfers to JOIN when all paths have been started. B4 starts a new parallel path. B5 is a parallel path. B6 is the JOIN.

In contrast to the examples given by Opler and Anderson, this scheme is independent of the number of processors available, and there is no need to state

Example 2. A PARALLEL FOR



explicitly the relationship between the FORK and the JOIN.

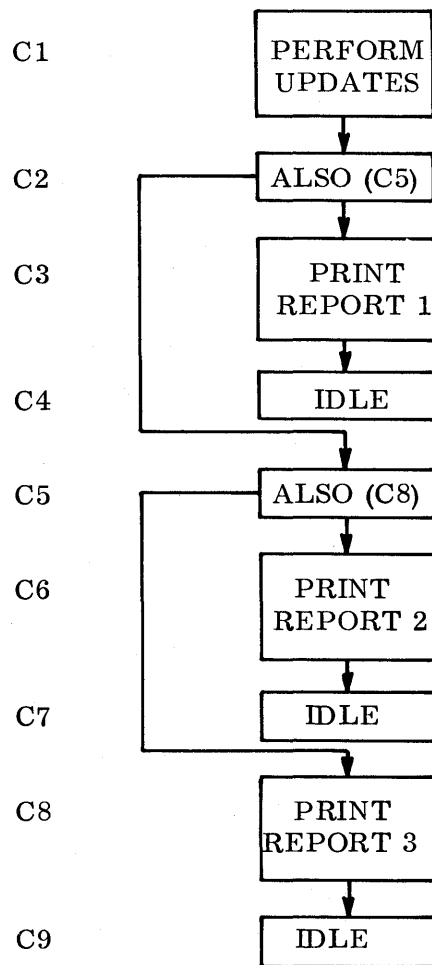
This process allows the number of paths to be data dependent, and even to be zero. Note that the PPC shows how many parallel paths are active. This concept has been described and defined in ALGOL by Wirth.⁵⁸ He uses the term AND (which he attributes to van Wijngaarden, and which we borrow for consistency) but he does not show how the system controls the JOIN nor does he provide an explicit PARALLEL FOR. The programmer builds it himself. Wirth is correct in that a programming language does not specifically require the JOIN but the hardware does need a JOIN delimiter.

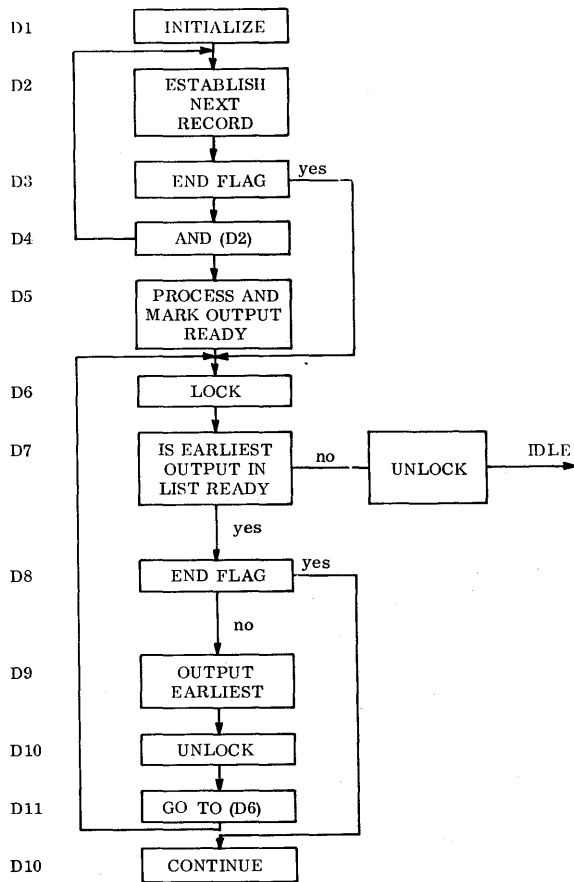
Example 3 shows how unjoined forks are implemented, which is the example discussed in Item 5 under Parallel Structures. Note that PREP is not used, no PPC counters are involved, and IDLE and ALSO are used instead of JOIN and AND.

Example 4 shows how the classic file maintenance, or payroll, application can be handled. The preliminary part is similar to the PARALLEL FOR but the JOIN structure is different because we take the case where the output is required to be in the same sequence as the input. To do this, each record processed has a link set to point to the following record. This is set when the following record is established in D2.

D1 opens files, sets initial links, and sets data areas. D2 gets the next record and makes links. D3 ends the loop that produces the parallel paths; D4 produces the parallel paths; and D5 is the parallel path. The JOIN occurs at D6 and uses a lockout mechanism to ensure that only one processor executes output at a time and in proper sequence. Where parallel processing is allowed, there are some functions that must be carried out by only one proc-

Example 3. Unjoined Forks



Example 4. Serial File Processing

ess at a time, for example, updating a record. Various hardware locks have been developed,³ and even some complex software locks.^{22,35} Wirth⁵⁸ proposed a term **SHARED** in ALGOL to denote procedures that are not to be used in parallel.

Special Points to Note

First, we must note that these techniques presuppose that all programs use re-entrant code.

Second, we must provide some special protected functions such as “add to x” so that totals can be accumulated from each path, or a lock-out feature to temporarily restrict access to a section of data.^{2,3,22,35,43,59}

Third, we must arrange the linkage in Example 4 so that the system can tolerate such situations as the case of an empty file, and one record being completed before the next is initiated.

This is necessary because the link from one record to its successor, used to control output, may

not be established if a delay occurs such that a record is output before its successor is input. One alternative is to delay output; another is to provide an interim dummy linkage.

Fourth, we must make box D2 include choosing an insertion to the file. The full logic for this box determines whether the next record to be processed is

1. one record from the master file and a matching record from the detail file.
2. one record from the master file and no matching detail record.
3. one record from the detail file and no matching master record.

PROCESSOR SCHEDULING

Apart from the wish to run certain problems faster in a multiprocessor, the most interesting case in scheduling is the possibility of shared access to program segments by processors and the subsequent reduction in overlay overheads. It may seem at first that this would lead to more complex dynamic storage allocation than is desirable. However, it turns out that shared access arises naturally. There are three basic advantages that this scheme enjoys:

- Processes generate parallel paths dynamically one at a time on an “as needed” basis, which contrasts with the schemes proposed by Opler⁴⁵ and Anderson.² It limits scheduling overheads in that the process generates only records of parallel processes as they are encountered. For example, the number existing for an N-way **PARALLEL-FOR** will be only one plus the number initiated, however large N may be.
- Distribution of parallel paths among varying numbers of processors is an easy ad hoc process.
- The same basic data can be used by the scheduler to exploit any of the alternative advantages of parallel processing.

The following examples illustrate the last two points. Consider a system with three processors: I, II, and III. Consider that there are six tasks in the system. Each task has one or more processes live at a time. Each process is either active in a processor or is in one of three queues. A process in queue ‘a’ has the segments it needs in core, but no processor.

Table I
An example without priorities.

EVENT	STATUS RESULT																								
Five processes exist, A1 through E1 Four have segments in core, A1 through D1 Three are active, A1 through C1	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>I</td><td>II</td><td>III</td><td>a</td><td>c</td></tr> </table>	Process	A1	B1	C1	D1	E1	Segment	S1	T1	U1	V1	W1	State	I	II	III	a	c						
Process	A1	B1	C1	D1	E1																				
Segment	S1	T1	U1	V1	W1																				
State	I	II	III	a	c																				
A1 creates A2 which uses S1, and A2 goes into queue 'a'	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td><td>A2</td></tr> <tr><td>Segment</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td><td>S1</td></tr> <tr><td>State</td><td>I</td><td>II</td><td>III</td><td>a</td><td>c</td><td>a</td></tr> </table>	Process	A1	B1	C1	D1	E1	A2	Segment	S1	T1	U1	V1	W1	S1	State	I	II	III	a	c	a			
Process	A1	B1	C1	D1	E1	A2																			
Segment	S1	T1	U1	V1	W1	S1																			
State	I	II	III	a	c	a																			
A1 needs next segment S2, A1 goes into queue 'c', D1 obtains processor I	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td><td>A2</td></tr> <tr><td>Segment</td><td>S2</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td><td>S1</td></tr> <tr><td>State</td><td>c</td><td>II</td><td>III</td><td>I</td><td>c</td><td>a</td></tr> </table>	Process	A1	B1	C1	D1	E1	A2	Segment	S2	T1	U1	V1	W1	S1	State	c	II	III	I	c	a			
Process	A1	B1	C1	D1	E1	A2																			
Segment	S2	T1	U1	V1	W1	S1																			
State	c	II	III	I	c	a																			
B1 needs next segment T2, B1 goes into queue 'c', A2 obtains processor II	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td><td>A2</td></tr> <tr><td>Segment</td><td>S2</td><td>T2</td><td>T1</td><td>V1</td><td>W1</td><td>S1</td></tr> <tr><td>State</td><td>c</td><td>c</td><td>III</td><td>I</td><td>c</td><td>II</td></tr> </table>	Process	A1	B1	C1	D1	E1	A2	Segment	S2	T2	T1	V1	W1	S1	State	c	c	III	I	c	II			
Process	A1	B1	C1	D1	E1	A2																			
Segment	S2	T2	T1	V1	W1	S1																			
State	c	c	III	I	c	II																			
Now a new segment can be called to overlay T1. W1 has waited longest and E1 goes into 'b'	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td><td>A2</td><td>A3</td></tr> <tr><td>Segment</td><td>S2</td><td>T2</td><td>U1</td><td>V1</td><td>W1</td><td>S1</td><td>S1</td></tr> <tr><td>State</td><td>c</td><td>c</td><td>III</td><td>I</td><td>b</td><td>II</td><td>a</td></tr> </table>	Process	A1	B1	C1	D1	E1	A2	A3	Segment	S2	T2	U1	V1	W1	S1	S1	State	c	c	III	I	b	II	a
Process	A1	B1	C1	D1	E1	A2	A3																		
Segment	S2	T2	U1	V1	W1	S1	S1																		
State	c	c	III	I	b	II	a																		
A2 needs next segment S2, A2 goes into queue 'c', A3 obtains processor II	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td><td>A2</td><td>A3</td></tr> <tr><td>Segment</td><td>S2</td><td>T2</td><td>U1</td><td>D1</td><td>W1</td><td>S2</td><td>S1</td></tr> <tr><td>State</td><td>c</td><td>c</td><td>III</td><td>I</td><td>b</td><td>c</td><td>II</td></tr> </table>	Process	A1	B1	C1	D1	E1	A2	A3	Segment	S2	T2	U1	D1	W1	S2	S1	State	c	c	III	I	b	c	II
Process	A1	B1	C1	D1	E1	A2	A3																		
Segment	S2	T2	U1	D1	W1	S2	S1																		
State	c	c	III	I	b	c	II																		
W1 arrives and E1 goes into queue 'a'	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td><td>A2</td><td>A3</td></tr> <tr><td>Segment</td><td>S2</td><td>T2</td><td>U1</td><td>D1</td><td>W1</td><td>S2</td><td>S1</td></tr> <tr><td>State</td><td>c</td><td>c</td><td>III</td><td>I</td><td>b</td><td>c</td><td>II</td></tr> </table>	Process	A1	B1	C1	D1	E1	A2	A3	Segment	S2	T2	U1	D1	W1	S2	S1	State	c	c	III	I	b	c	II
Process	A1	B1	C1	D1	E1	A2	A3																		
Segment	S2	T2	U1	D1	W1	S2	S1																		
State	c	c	III	I	b	c	II																		
C1 needs next segment U1. C1 goes into queue 'c'. E1 obtains III	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td><td>A2</td><td>A3</td></tr> <tr><td>Segment</td><td>S2</td><td>T2</td><td>U1</td><td>D1</td><td>W1</td><td>S2</td><td>S1</td></tr> <tr><td>State</td><td>c</td><td>c</td><td>c</td><td>I</td><td>III</td><td>c</td><td>II</td></tr> </table>	Process	A1	B1	C1	D1	E1	A2	A3	Segment	S2	T2	U1	D1	W1	S2	S1	State	c	c	c	I	III	c	II
Process	A1	B1	C1	D1	E1	A2	A3																		
Segment	S2	T2	U1	D1	W1	S2	S1																		
State	c	c	c	I	III	c	II																		
Now U1 can be overlaid. S2 is chosen because two processes in queue 'c' need it. A1 and A2 go into queue 'b'.	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td><td>A2</td><td>A3</td></tr> <tr><td>Segment</td><td>S2</td><td>T2</td><td>U1</td><td>D1</td><td>W1</td><td>S2</td><td>S1</td></tr> <tr><td>State</td><td>b</td><td>c</td><td>c</td><td>I</td><td>III</td><td>b</td><td>II</td></tr> </table>	Process	A1	B1	C1	D1	E1	A2	A3	Segment	S2	T2	U1	D1	W1	S2	S1	State	b	c	c	I	III	b	II
Process	A1	B1	C1	D1	E1	A2	A3																		
Segment	S2	T2	U1	D1	W1	S2	S1																		
State	b	c	c	I	III	b	II																		

A process in queue 'c' does not have all its segments in core. A process in queue 'b' is awaiting termination of some peripheral I/O, in particular, perhaps the arrival of its needed segment in core.

The state of the system can be represented by Table I. The tasks are labelled A1 through E1. When they fork into separate processes, numeric tags are used. For simplicity we assume that any process needs only one segment at a time and there is only room for four overlay slots in executable storage (i.e., storage from which processors select and execute instructions). It is obviously desirable to have more overlay slots than processors to build up work for processors. The current segment needed by a process is shown in line 2. The status of each process is shown as either the identity of the processor executing it or the queue 'a', 'b', or 'c' where it is located.

In the case of equal priority processes we assert that the scheduler (i) prefers to initiate processes in queue 'a', (ii) prefers to get a segment wanted by most members of 'b', and (iii) gives chronological preference.

Table I shows how the system might react with a set of equal priority processes and how a set of par-

allel paths builds up demand for its segments. Table II shows how the system reacts for priority classes and uses low priorities to fill the gaps.

ASSETS OF THIS SCHEME

There are nine significant assets of this way of considering parallel processing:

1. It indicates profitable sources of parallelism in conventional program structures.
2. It enables some parallelism in FOR statements to be shown in existing programs with trivial changes.
3. It provides an easy, and hopefully popular, way to express parallelisms in all conventional programming languages.
4. It provides a simple mechanism to control parallel activities which could also be incorporated in hardware.
5. It requires only a small expansion to the bookkeeping required in operating systems.

Table II
An example with priorities.

EVENT	STATUS RESULT																								
Five processes exist, A1 through E1 Four have segments in core, A1 through D1 Three are active, A1 through C1	<table border="1"> <tr><td>Process</td><td>A1</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>I</td><td>II</td><td>III</td><td>a</td><td>c</td></tr> </table>	Process	A1	B1	C1	D1	E1	Segment	S1	T1	U1	V1	W1	State	I	II	III	a	c						
Process	A1	B1	C1	D1	E1																				
Segment	S1	T1	U1	V1	W1																				
State	I	II	III	a	c																				
A1 creates A2, which uses S1 and goes into queue 'a'	<table border="1"> <tr><td>Process</td><td>A1</td><td>A2</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S1</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>I</td><td>a</td><td>II</td><td>III</td><td>a</td><td>c</td></tr> </table>	Process	A1	A2	B1	C1	D1	E1	Segment	S1	S1	T1	U1	V1	W1	State	I	a	II	III	a	c			
Process	A1	A2	B1	C1	D1	E1																			
Segment	S1	S1	T1	U1	V1	W1																			
State	I	a	II	III	a	c																			
A2 preempts C1 and gets III, C1 goes into queue 'a'	<table border="1"> <tr><td>Process</td><td>A1</td><td>A2</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S1</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>I</td><td>III</td><td>II</td><td>a</td><td>a</td><td>c</td></tr> </table>	Process	A1	A2	B1	C1	D1	E1	Segment	S1	S1	T1	U1	V1	W1	State	I	III	II	a	a	c			
Process	A1	A2	B1	C1	D1	E1																			
Segment	S1	S1	T1	U1	V1	W1																			
State	I	III	II	a	a	c																			
A1 needs next segment S2 and goes into queue 'c'. C1 gets I	<table border="1"> <tr><td>Process</td><td>A1</td><td>A2</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S2</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>c</td><td>III</td><td>II</td><td>I</td><td>a</td><td>c</td></tr> </table>	Process	A1	A2	B1	C1	D1	E1	Segment	S2	S1	T1	U1	V1	W1	State	c	III	II	I	a	c			
Process	A1	A2	B1	C1	D1	E1																			
Segment	S2	S1	T1	U1	V1	W1																			
State	c	III	II	I	a	c																			
Priority calls for S2 to be requested to overwrite V1. A1 goes into queue 'b', D1 goes into queue 'c'	<table border="1"> <tr><td>Process</td><td>A1</td><td>A2</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S2</td><td>S1</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>b</td><td>III</td><td>II</td><td>I</td><td>c</td><td>c</td></tr> </table>	Process	A1	A2	B1	C1	D1	E1	Segment	S2	S1	S1	T1	U1	V1	W1	State	b	III	II	I	c	c		
Process	A1	A2	B1	C1	D1	E1																			
Segment	S2	S1	S1	T1	U1	V1	W1																		
State	b	III	II	I	c	c																			
A2 creates A3 which uses S1 and goes into queue 'a'	<table border="1"> <tr><td>Process</td><td>A1</td><td>A2</td><td>A3</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S2</td><td>S1</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>b</td><td>IV</td><td>a</td><td>II</td><td>I</td><td>c</td><td>c</td></tr> </table>	Process	A1	A2	A3	B1	C1	D1	E1	Segment	S2	S1	S1	T1	U1	V1	W1	State	b	IV	a	II	I	c	c
Process	A1	A2	A3	B1	C1	D1	E1																		
Segment	S2	S1	S1	T1	U1	V1	W1																		
State	b	IV	a	II	I	c	c																		
A3 preempts C1 and gets I, C1 goes into queue 'a'	<table border="1"> <tr><td>Process</td><td>A1</td><td>A2</td><td>A3</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S2</td><td>S1</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>b</td><td>III</td><td>I</td><td>II</td><td>a</td><td>c</td><td>c</td></tr> </table>	Process	A1	A2	A3	B1	C1	D1	E1	Segment	S2	S1	S1	T1	U1	V1	W1	State	b	III	I	II	a	c	c
Process	A1	A2	A3	B1	C1	D1	E1																		
Segment	S2	S1	S1	T1	U1	V1	W1																		
State	b	III	I	II	a	c	c																		
S2 overwrites V1, A1 goes into queue 'a'	<table border="1"> <tr><td>Process</td><td>A1</td><td>A2</td><td>A3</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S2</td><td>S1</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>a</td><td>III</td><td>I</td><td>II</td><td>a</td><td>c</td><td>c</td></tr> </table>	Process	A1	A2	A3	B1	C1	D1	E1	Segment	S2	S1	S1	T1	U1	V1	W1	State	a	III	I	II	a	c	c
Process	A1	A2	A3	B1	C1	D1	E1																		
Segment	S2	S1	S1	T1	U1	V1	W1																		
State	a	III	I	II	a	c	c																		
A1 preempts B1 and gets II B1 goes into queue 'a'	<table border="1"> <tr><td>Process</td><td>A1</td><td>A2</td><td>A3</td><td>B1</td><td>C1</td><td>D1</td><td>E1</td></tr> <tr><td>Segment</td><td>S2</td><td>S1</td><td>S1</td><td>T1</td><td>U1</td><td>V1</td><td>W1</td></tr> <tr><td>State</td><td>II</td><td>III</td><td>I</td><td>a</td><td>a</td><td>c</td><td>c</td></tr> </table>	Process	A1	A2	A3	B1	C1	D1	E1	Segment	S2	S1	S1	T1	U1	V1	W1	State	II	III	I	a	a	c	c
Process	A1	A2	A3	B1	C1	D1	E1																		
Segment	S2	S1	S1	T1	U1	V1	W1																		
State	II	III	I	a	a	c	c																		

6. It is applicable to all types of computer structures and applications.
7. It permits the dynamic changing of the number of processors available.
8. It permits the exploiting of a re-entrant code by individual programs.
9. It offers opportunities to reduce heavy page turning overheads.

REFERENCES—BIBLIOGRAPHY

1. G. M. Amdahl, "New Concepts In Computing System Design," *Proc. IRE*, vol. 50, (May 1962) pp. 1073-1077.
2. J. P. Anderson, "Program Structures for Parallel Processing," *CACM*, vol. 8, (December 1965) pp. 786-788.
3. J. P. Anderson, et al, "D 825—A Multiple-Computer System for Command and Control," *AFIPS, volume 22, Proc. FJCC*, Spartan Books, Washington, D.C., 1962, pp. 86-96.
4. J. R. Ball, et al, "On the Use of the SOLOMON Parallel-Processing Computer," *AFIPS, volume 22, Proc. FJCC*, Spartan Books, Washington, D.C., 1962, p. 137.
5. G. P. Bergin, "Method of Control for Re-entrant Routines," *AFIPS, volume 26, Proc. FJCC*, Spartan Books, Washington, D.C., 1964, pp. 45-55.
6. S. Boilen, et al, "A Time-Sharing Debugging System for a Small Computer," *AFIPS, volume 23, Proc. SJCC*, Spartan Books, Washington, D.C., 1963, p. 51.
7. F. P. Brooks, Jr., "The Future of Computer Architecture," *Proc. IFIP*, volume 1, 1965, p. 87.
8. E. Bryan, "The Dynamic Characteristics of Computer Programs," Rand Corporation, 1964 (Paper to SHARE XXII).
9. W. Buckholz (Ed.), *Planning a Computer System*, McGraw-Hill, New York, 1962.
10. A. W. Burks, *The Logic of Fixed and Growing Automata*, University of Michigan, Eng. Res. Inst., Ann Arbor, 1957.
11. W. Chang and D. J. Wong, "Analysis of Real-Time Programming," *Journal ACM*, vol. 12, (1965) p. 581.
12. T. E. Cheatham and G. F. Leonard, "An Introduction to the CL-II Programming System," Computer Associates-63-7-SD, November 1963.
13. E. F. Codd, "Multiprogram Scheduling," *CACM*, (June & July 1963) pp. 347 and 413.
14. E. F. Codd, "Multiprogramming STRETCH: Feasibility Considerations," *CACM*, vol. 2, (November 1959) pp. 13-17.
15. E. G. Coffman, Jr., et al, "A General-Purpose Time-Sharing System," *AFIPS, volume 25, Proc. SJCC*, Spartan Books, Washington, D.C., 1964.
16. M. Phyllis Cole, et al, "Operational Software in a Disc Oriented System," *AFIPS, volume 26, Proc. FJCC*, Spartan Books, Washington, D.C., 1964, p. 351.
17. M. E. Conway, "A Multi-Processor System Design," *AFIPS, volume 24, Proc. FJCC*, Spartan Books, Washington, D.C., 1963, pp. 139-146.
18. M. E. Conway, "Design of a Separate Transition Diagram Compiler," *CACM*, vol. 6, (July 1963) pp. 396-408.
19. F. J. Corbató, et al, "An Experimental Time-Sharing System," *Proc. SJCC*, vol. 21, (1962) pp. 335-344.
20. J. B. Dennis, "Segmentation and the Design of Multi-Programmed Computer Systems," *Journal ACM*, vol. 12, (Oct. 1965) p. 589.
21. E. W. Dijkstra, "Solution of a Problem in Concurrent Programming Control," *CACM*, vol. 8, (Sept. 1965) p. 569.
22. Letter: "A Problem in Concurrent Programming Control," *CACM*, vol. 8, (Sept. 1965) p. 569.
23. P. Dreyfus, "System Design of the Gamma 60," *Proc. WJCC*, 1958, p. 130.
24. S. Fernback, "Computers in the USA—Today and Tomorrow," *Proc. IFIP*, vol. 1, 1965, p. 77.
25. D. R. Fitzwater and E. J. Schweppe, "Consequent Procedures in Conventional Computers," *AFIPS, volume 26, Proc. FJCC*, Spartan Books, Washington, D.C., 1964, pp. 465-76.
26. S. Gill, "Parallel Programming," *Computer Journal*, vol. 1, (1958) p. 2.
27. A. C. D. Haley, "The KDF. 9. Computer System," *AFIPS, volume 22, Proc. FJCC*, Spartan Books, Washington, D.C., 1962, p. 108.
28. J. H. Holland, "The Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously," *Proc. EJCC*, 1959, p. 108-113.
29. J. H. Holland, "Iterative Circuit Computers," *Proc. WJCC*, 1960, p. 259.
30. T. J. Howarth, et al, "The Manchester University Atlas Operating System, Part II: User's Description," *Computer Journal*, vol. 4, (1961) p. 226.
31. D. J. Howarth, "Experience with the Atlas Scheduling System," *AFIPS, volume 23, Proc. SJCC*, Spartan Books, Washington, D.C., 1963, p. 59.
32. E. T. Irons, "A Rapid Turnaround Multi-Programming System," *CACM*, vol. 8, (March 1965) p. 152-7.
33. T. Kilburn, et al, "The Manchester University Atlas Operating System, Part I: Internal Organization," *Computer Journal*, vol. 4, (1961) p. 222.

34. T. Kilburn, et al, "The Atlas Supervisor," *Proc. EJCC*, vol. 20, 1961, pp. 279-294.
35. D. E. Knuth, Letter: "Additional Comments on a Problem in Concurrent Programming Control," *CACM*, vol. 9, (May 1966) p. 321.
36. N. Landis, et al, "Initial Experience with an Operating Multi-Programming System," *CACM*, vol. 5, (May, 1962) pp. 282-286.
37. A. L. Leiner, et al, "Concurrently Operating Computer Systems," *ICIP*, 1959, B. 12. 17. pp. 353-361.
38. A. L. Leiner, et al, "Organizing a Network of Computers to Meet Deadlines," *Proc. EJCC*, 1957, pp. 115-128.
39. G. F. Leonard and J. R. Goodroe, "An Environment for an Operating System," *ACM Proc.* 64, E. 2.3-1.
40. F. M. Marcotty, et al, "Time-Sharing on the Ferranti-Packard F. P. Good Computer System," *AFIPS, volume 23, Proc. SJCC*, Spartan Books, Washington, D.C., 1963, p. 29.
41. J. L. McKenney, "Simultaneous Processing of Jobs on an Electronic Computer," *Management Sci.* vol. 8, (April, 1962) pp. 344-54.
42. M. R. Mills, "Operational Experience of Time Sharing and Parallel Processing," *Computer Journal*, vol. 6, (1963) p. 28.
43. M. R. Nekora, "Comment on a Paper on Parallel Processing," *CACM*, vol. 4, (Feb. 1961) p. 103.
44. J. Nievergelt, "Parallel Methods for Integrating Ordinary Differential Equations," *CACM*, vol. 7, (Dec. 1964) pp. 731-33.
45. A. Opler, "Procedure Oriented Language Statements to Facilitate Parallel Processing," *CACM*, vol. 8, (May 1965) pp. 306-7.
46. G. E. Pickering, et al, "Multi-Computer Programming for a Large Scale Real-Time Data Processing System," *AFIPS, volume 25, Proc. SJCC*, Spartan Books, Washington, D.C., 1964, p. 445.
47. R. E. Porter, (Title unknown) Parallel Programming Conference, May 1960.
48. Jeanne Poyen, Private discussion with Opler, 1959.
49. G. Radin and H. P. Rogoway, "NPL: Highlights of a New Programming Language," *CACM*, vol. 8, (Jan. 1965) pp. 9-17.
50. M. R. Rosenberg, "Computer-Usage Accounting for Generalized Time-Sharing Systems," *CACM*, vol. 7, (May 1964) pp. 304-8.
51. W. F. Schmitt and A. B. Tonik, "Sympathetically Programmed Computers," *ICIP*, 1959, pp. 344-348.
52. E. S. Schwartz, "An Automatic Sequencing Procedure with Application to Parallel Programming," *JACM*, vol. 8, (April 1961) p. 513.
53. A. B. Shafritz, et al, "Multi-Level Programming for a Real-Time System," *Proc. EJCC*, vol. 20, 1961, pp. 1-16.
54. D. L. Slotnick, et al, "The Solomon Computer," *AFIPS, volume 22, Proc. FJCC*, Spartan Books, Washington, D.C., 1962, p. 97.
55. J. S. Squire and Sandra M. Palaiss, "Programming and Design Consideration of a Highly Parallel Computer," *AFIPS, volume 23, Proc. SJCC*, Spartan Books, Washington, D.C., 1963, p. 395.
56. E. Strachey, "Time-Sharing in Large Fast Computers," *ICIP*, 1959, p. 336 and *Computers & Automation*, (Aug. 1959).
57. R. N. Thompson and J. A. Wilkinson, "The D825 Automatic Operating and Scheduling System," *AFIPS, volume 23, Proc. SJCC*, Spartan Books, Washington, D.C., 1963, p. 41.
58. N. Wirth, Letter: "A Note on Program Structures for Parallel Processing," *CACM*, vol. 9, (May 1966) p. 320.
59. L. D. Yarborough, "Some Thoughts on Parallel Processing," *CACM*, vol. 3, (Oct. 1960) p. 539.

THE LISP 2 PROGRAMMING LANGUAGE AND SYSTEM *

Paul W. Abrahams

Information International, Inc., New York

Jeffrey A. Barnett, Erwin Book, Donna Firth,
Stanley L. Kameny, Clark Weissman

System Development Corporation, Santa Monica, California

and

Lowell Hawkinson, Michael I. Levin, Robert A. Saunders

Information International, Inc., Los Angeles, California

INTRODUCTION

LISP 2 is a new programming language designed for use in problems that require manipulation of highly complex data structures as well as lengthy arithmetic operations. Presently implemented on the AN/FSQ-32V computer at the System Development Corporation in Santa Monica, California, LISP 2 has two components: the language itself, and the programming system in which it is embedded. The system programs that define the language are accessible to and modifiable by the user; thus the user has an unparalleled ability to shape the language to suit his own needs and to utilize parts of the system as building blocks in constructing his own programs.

While it provides these capabilities to the do-it-yourself programmer, LISP 2 also provides the com-

plete and convenient programming facilities of a ready-made system. Typical application areas for LISP 2 include heuristic programming, algebraic manipulation, linguistic analysis and machine translation of natural and artificial languages, analysis of particle reactions in high-energy physics, artificial intelligence, pattern recognition, mathematical logic and automata theory, automatic theorem proving, game-playing, information retrieval, numerical computation, and exploration of new programming technology.

The primary source materials on LISP 2 are the LISP 2 Primer,¹ which provides an introduction to the language for those with little or no programming experience, and the LISP 2 Reference Manual,² which provides a complete specification of the language.

The LISP 2 programming system provides not only a compiler, but also a large collection of run-time facilities. These facilities include the library functions, a monitor for control and on-line interac-

* Produced by SDC and III in performance of contract AF 19(628)-5166 with the Electronic Systems Division, Air Force Systems Command, in performance of ARPA Order 773 for the Advanced Research Projects Agency, Information Processing Techniques Office, and Subcontract 65-107.

tion, automatic storage management, and communication with the monitor system of the machine on which the system is operating.

A particularly important part of the program library is a group of programs for bootstrapping LISP 2 onto a new machine. (Bootstrapping is the standard method for creating a LISP 2 system on a new machine.) The bootstrapping capability is sufficiently powerful so that the new machine requires no resident programs other than the standard monitor system and a binary loader.

LISP 2 includes and extends the capabilities of its ancestor, LISP 1.5.³ LISP 1.5 has been notable for its mathematical elegance and symbol-manipulating capabilities. It is unique among programming languages in the ease with which programs can be treated as data, in its "garbage collection" approach to reclaiming unused storage, and in its ability to represent programs organized as a collection of small, easily understood function definitions. Full recursion without special user provisions is a natural outgrowth of the structure of the language. However, LISP 1.5 lacks a convenient input language and efficiency in the treatment of purely arithmetic operations.

LISP 2 was designed to maintain the advantages of LISP 1.5 while remedying its deficiencies. The first major change has been the introduction of two distinct language levels: Source Language (SL) and Intermediate Language (IL). The two languages have different syntaxes but the same semantics (in the sense that for every SL program there is a computationally equivalent IL program). The syntax of SL resembles that of ALGOL 60,⁴ while the syntax of IL resembles that of LISP 1.5. IL is designed to have the same structure as data, and thus to be capable of being manipulated easily by user (and system) programs. An advantage of the ALGOL-like source language is that the ALGOL algorithms can be utilized with little change.

The second major change has been the introduction of type declarations and new data types, including integer-indexed arrays and character strings. At a future time, packed data tables, which can presently be simulated through programming techniques, will be added. Type declarations are necessary to obtain efficient compiled code, particularly for arithmetic operations, but by using the default mechanisms, a programmer may omit type declarations entirely (albeit at the cost of efficiency).

The third major change has been the introduction of partial-word extraction and insertion operators. Further, an IL-level macro expansion capability has

been included, which makes possible the definition of operations in terms of a basic set of open-coded primitives. These changes made it possible to write the entire system in its own language without loss of efficiency. At the same time, the compilations of user programs are more economical in time, and to some extent in space, than they would be without these facilities. Furthermore, the knowledgeable user can trade space against time through appropriate redefinition of system functions.

A fourth major change, the introduction of pattern-driven data manipulation facilities, along the lines of COMIT⁵ and METEOR,⁶ is still in the process of implementation. Because of the open-ended nature of LISP 2, these facilities can be added without disrupting the existing system structure. We mention this facility here, despite the fact that it does not yet exist, because it is an integral part of the over-all design of the language. Since the specifications are not final as of this writing, however, we shall not discuss them further.

To orient the reader toward the exposition of the language, we present a short example at this point. Further examples will be given later. The following program⁷ is written in SL:

```
% RANDOM COMPUTES A RANDOM
  NUMBER IN THE INTERVAL (A, B)
  OWN INTEGER Y;
  REAL FUNCTION RANDOM(A,B);
    REAL A,B;
  BEGIN Y←3125*Y;
        Y←Y\67108864;
        RETURN (Y/67108864.0 * (B-
          A)+A)
  END;
```

The only significant difference between this program and the ALGOL original is the use of the reverse slash "\ " to indicate the computation of the remainder. The corresponding program in IL is:

```
(DECLARE (Y OWN INTEGER))
(FUNCTION (RANDOM REAL)
 ((A REAL) (B REAL))
 (BLOCK NIL (SET Y (TIMES 3125 Y))
 (SET Y (REMAINDER Y 67108864))
 (RETURN (PLUS (TIMES (QUOTIENT
 Y 6.7108864000E+7)
 (DIFFERENCE B A )) A))))
```

The process of converting SL programs into compiled code is shown in Fig. 1. SL is first translated into IL by syntax translator. IL is then translated

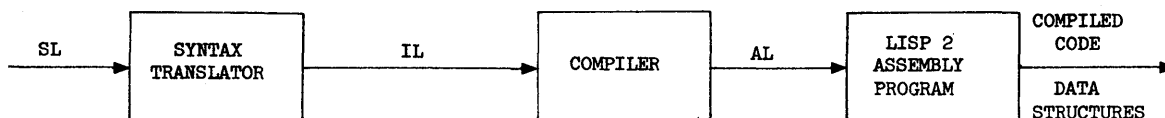


Figure 1. System organization. SL = source language; IL = intermediate language; AL = assembly language.

into assembly language by a compiler. Finally, the assembly language is translated into machine language by an assembly program. The process is entirely accessible to the user, in that he can write programs in IL or assembly language if he so chooses.

The remainder of this paper is divided into two parts, one dealing with the language and the other with the implementation. Certain aspects of the language that were intended primarily as implementation tools, e.g., open subroutines, are discussed in connection with the implementation.

In discussing the language, we shall present simultaneous discussions of the syntax of SL and IL, accompanied by discussion of the semantics of both. In this way the semantic equivalence of SL and IL will become apparent. It should be borne in mind that the primary use of SL is for programs written by people, while the primary use of IL is for programs written by machines. Thus the syntax of SL is designed for convenience in writing, while the syntax of IL is designed to reflect in its form the structure of the program that it represents.

THE LISP 2 LANGUAGE

Tokens

Tokens are the smallest units of input or output data with which LISP 2 programs ordinarily deal and are significant because of their role in defining the standard input/output conventions with regard to both programs and data. The major categories of tokens are:

1. Delimiters
2. Numbers
3. Simple strings
4. Identifiers
5. Operators

The delimiter tokens are:

() [] cr

Numbers as tokens may be either signed or unsigned in IL, but must be unsigned in SL since a preceding sign is interpreted as an operator. Some examples of unsigned numbers are:

```

unsigned
integer  1      2      3E5
unsigned
octal    12Q    14Q6
unsigned
real     .87    12.    4.5E5    2.E-10
  
```

Signed numbers are like these, but are preceded by a sign. Other examples of tokens are:

```

identifier  AB    H21    GO.TO
operator    * /  = ↑ > = \ + ←
  
```

A string consists of a sequence of characters delimited at each end by "#". The character "'" inside a string causes the character following to be entered in the string. Some examples of strings are:

```

#AB(C)D#
#A'#256'# #
#ISN'T#
  
```

An identifier may be created from a string by preceding it with the escape character. This character is changeable within the system but will usually be "%." If "%" is the escape character, the following is an identifier:

```
% #AB(C)D#
```

An identifier created in this way is said to have an "unusual spelling," since, in general, such identifiers will be created only when they cannot be written in any other way unambiguously.

Data

The most general form of a LISP 2 datum is an S-expression, where the S stands for "symbolic." S-expressions are built up from atoms, which may be numbers, strings, identifiers, function specifiers, and arrays. As in LISP 1.5, the class of S-expressions is defined recursively as follows:

1. Every atom is an S-expression.
2. If e_1 and e_2 are S-expressions, then

$(e_1 . e_2)$

is an S-expression. Thus, for instance,

((A . B) . (C . D))

is an S-expression.

S-expressions of the form:

(e₁ . (e₂ (e_n . NIL) . . .))

are known as lists, and can be written in the abbreviated form:

(e₁ e₂ . . . e_n)

The e_i are called the elements of the list. The two notations may be intermixed; thus

((A . 1) (B . 2) . . . (Z . 26))

is an S-expression in the form of a list, but the elements of the list are not themselves in the form of lists. The atom NIL can also be written in the form (), and designates the empty list.

The LISP functions CAR, CDR, and CONS are defined by:

CAR applied to (e₁ . e₂) yields e₁

CDR applied to (e₁ . e₂) yields e₂

CONS applied to e₁ and e₂ yields (e₁ . e₂)

In terms of the list notation, CAR finds the first element of a list and CDR removes the first element from a list. Thus CAR applied to the list (A B C D) yields A, and CDR applied to the same list yields the list (B C D). CDR applied to a list of one element yields the empty list (). The function NULL has value TRUE for the empty list () (also represented as NIL) and value FALSE for anything else. The function CONS of two arguments can be used to add an element at the head of a list; thus CONS applied to the element A and the list (B C D) yields the list (A B C D). CONS is the basic operator used for constructing lists.

IL programs are written in the form of S-expressions, and therefore can be treated as data. The ability to treat programs as data in a natural way is an essential feature of LISP. SL programs can also be treated as data, because of the existence of strings; however, this is not nearly so natural as it is with IL.

Arrays are atoms because CAR and CDR are not defined for them. Constant arrays are written by enclosing their elements in brackets. For example:

[2 5 -1 4]

is a one-dimensional array of integers, and:

[[A B C] [A1 B1 C1] [A2 B2 C2] [A3 B3 C3]]

is a two-dimensional array of S-expressions.

Data Types. Although every LISP 2 datum is an S-expression, it is useful to pick out certain subsets of the set of all S-expressions and to designate these subsets by data type names. The data type names and the subsets they denote are:

BOOLEAN	Truth value data, represented by TRUE and FALSE. The empty list (), the atom NIL, and the Boolean value FALSE are regarded as synonymous.
INTEGER	Signed integers.
OCTAL	Another form of integer, basically regarded as unsigned, that prints in an octal output format.
REAL	Floating-point decimals.
FUNCTIONAL	LISP 2 function.
SYMBOL	The entire set of S-expressions. Strings and identifiers must be of this type.
type ARRAY	An array whose elements are of the specified type, where type is either BOOLEAN, INTEGER, OCTAL, REAL, FUNCTIONAL, or SYMBOL.

The different data types are not mutually exclusive, in that the class of data of type SYMBOL includes all other classes of data. Except for SYMBOL, all of the data classes include atomic data only.

Expressions

An expression is a designation of a datum. The datum designated by an expression is the value of the expression. The elementary components from which expressions are built up are constants, variables, and operational forms. We shall first discuss these, and then show how they are combined to form more complex expressions.

Constants. A constant is a datum appearing in a program context that denotes itself, i.e., its representation is both its name and its value. Consequently, a constant cannot change value during the execution of a program. A symbolic constant is denoted by a quoted S-expression. In SL, an S-expression is quoted by preceding it with a prime, e.g., 'ALPHA or '(L1 L2). In IL, an S-expression is quoted by preceding it with QUOTE in a list, e.g., (QUOTE ALPHA) or (QUOTE(L1 L2)). Quotation is necessary for identifiers and lists to prevent them from being interpreted as variables or operational forms.

Variables. A variable is also an elementary designation of a datum. However, the value of a variable may be changed during the execution of a program. A variable is normally denoted by a single identifier. Associated with every variable is a collection of bindings, each of which is a location containing a value. Bindings are created by declarations, which may appear in blocks, in functions, or on the supervisor level (see below). Blocks and functions are the two different kinds of program units. At execution time, a program unit may be activated either by the supervisor or by another program unit; thus there is a hierarchy of active program units.

When execution of a program unit commences, a binding is created for each variable declared by the program unit. When execution of the program unit is completed, these bindings disappear. Thus, each active program unit has a set of bindings associated with it, and the hierarchy of bindings corresponds to the hierarchy of active program units. In general, the value of a variable is the value attached to the most recently created and still existing binding of that variable. It is possible to use an assignment action to change the value associated with the current binding of a variable.

Associated with every variable is a type, a storage mode, and a transmission mode. The type of a variable restricts but does not necessarily determine the types of the data that are its values at different times. In particular, a variable whose type is *SYMBOL* may assume values of any type whatsoever.

There are three storage modes for variables: fluid, own, and lexical. A fluid variable can be referred to from outside the program unit that binds it, while a lexical variable cannot. Thus, fluid variables are more general but are also more prone to conflicts of names. Fluid variables are primarily used as a means of communication among separately compiled programs. An own variable is like a fluid variable except that only one binding can exist for it, and that binding must be made by a supervisor action. Own variables are designed primarily for communication with non-LISP 2 programs.

A variable may designate a datum either directly or indirectly. If the variable designates the datum directly, then it designates the actual value of the datum; if the variable designates the datum indirectly, then it designates the location in which the value is stored. This distinction is significant chiefly when a datum is being passed as an argument to a function; the transmission mode of the argument

variable indicates whether a value or a location of a value is being passed. If a location is being passed, then the transmission mode is said to be locative; otherwise the transmission mode is said to be by value.

Operational Forms. An operational form is used to apply a function to its arguments, to invoke a macro transformation, to alter the flow of a program, or to locate an element of an array. An operational form in SL is written:

$$f(e_1, e_2, \dots, e_n)$$

where f is the form operator and the e_i are its operands. In IL the operational form is written as:

$$(f e_1 e_2 \dots e_n)$$

If the form operator designates a function, then to obtain the value of the operational form, the operands are first evaluated, and then the function is applied to the values so obtained. An array is handled similarly; the subscripts are treated as arguments of a function that finds the desired element of the array.

Each function has associated with it a value type and a set of argument types. Any argument that is not of the expected type is converted to that type when the conversion is legal. The value type restricts the type of the result of the evaluation in the same way that the type of a variable restricts the values that the variable may assume.

In general, the order of evaluation of the operands of an operational form is not guaranteed. This is a departure from most other problem-oriented languages, but leads to improved compiled code. Also, with the advent of parallel processing computers it may be desirable to have several arguments evaluated simultaneously. If evaluating an operand has any side effect on the evaluation of any other operand, then the results of the evaluations will be unpredictable. However, the operator *ORDER* applied to an operational form will cause the operands to be evaluated in order of appearance.

Macros may be used to effect transformations of a program after it has been translated from SL to IL and before it has been compiled. When a macro name appears as a form operator, the effect at compile time is to cause the entire operational form to be replaced by a new form. The new form is calculated by a function associated with the macro; the argument of this function is the IL version of the operational form. Much of the task of compilation is

achieved through the use of macros that are invisible to the user; however, the user can also define his own macros. The use of macros is discussed further in connection with the user control facilities of the compiler.

Other Expressions. Elementary expressions (i.e., constants, variables, and operational forms) may be combined in SL by means of prefix and infix operators. Thus, all of the usual arithmetic and Boolean expressions are permitted in the usual algebraic notation. The symbolic operators CAR and CDR are also prefixes, which help to reduce the accumulation of parentheses. If a, b, and c are any expressions in SL, the relational expression:

$$a < b < c$$

and all similar forms have the same meaning in SL as they do in mathematics. Any number of relational operators can be combined in a relational expression, and different operators can be used in the same expression.

Infix and prefix operators cannot be used in IL, and must be replaced by corresponding operational forms. For example:

$A * B + 3 - \text{ALPHA} \uparrow 2$
is written in IL as:

(PLUS (TIMES A B) 3 (MINUS (EXPT ALPHA 2)))

A similar notation is used for relational expressions. Conditional expressions in SL have the form:

IF p_1 THEN e_1 ELSE IF p_2 THEN e_2 ELSE
... IF p_n THEN e_n ELSE e_{n+1}

The final ELSE clause need not be included (unlike ALGOL). The corresponding form in IL is:

(IF $p_1 e_1 p_2 e_2 \dots p_n e_n e_{n+1}$)

The p_i , which are Boolean expressions, are evaluated in turn from left to right until a true one is found. The value of the corresponding e_i is then used as the value of the entire expression. Conditional expressions have the useful property that evaluation proceeds only as far as necessary to determine the outcome.

A block expression is a block (see below) that appears in a context where an expression is required. A block expression is used to write a program as a sequence of statements to be executed and ultimately to produce a value. The value of a block is ordinarily

specified by a RETURN statement (see below). LISP 2 differs from ALGOL in permitting a block to be an expression as well as a statement.

A CASE expression is written in the form:

CASE(s, e_1, e_2, \dots, e_n)

in SL, and in the IL form:

(CASE $s e_1 e_2 \dots e_n$)

where s is an integer-valued expression known as the selector. If the value of the selector s lies in the range $1 \leq s \leq n$, then the expression e_s is evaluated and is the value of the CASE expression. If $s < 1$ or $s > n$, the value is e_n .

An assignment expression is written in the form:

$v \leftarrow e$

in SL and in the form:

(SET $v e$)

in IL.

If v is a variable and e is an expression, the assignment expression has the effect of evaluating the expression e and assigning its value to v . The value of the entire expression is the value of e . Assignment expressions, like all other expressions, can be used as arguments in operational forms; in particular, they can be nested to achieve simultaneous assignment of value to several variables.

The general form of the left side of an assignment expression is a locative expression. A locative expression designates a part of a data structure or variable structure. A variable is a particular case of a locative expression. Locative expressions can be used to designate the current binding of a variable, an element of an array, part of a list structure, or particular bits of a word of memory. Thus, the two assignments:

$A \leftarrow \text{'(MARY DOE) ;}$
 $\text{CAR } A \leftarrow \text{'JOHN ;}$

will cause the value of A to become:

(JOHN DOE)

Blocks and Statements

A block may be either a block expression, a block statement, or a compound statement. All three of these are written in the same form and are evaluated in the same way. Whether a block is a block expres-

sion, a block statement, or a compound statement depends on both the context of the block and what is contained within the block.

In SL, a block is written in the form:

```
BEGIN d1; d2; . . . dk; s1; s2; . . . sn END
```

where the d_i are block declarations and the s_i are statements. Each block declaration specifies one or more internal parameters, which are variables that are bound while the block is active. The corresponding form in IL is:

```
(BLOCK(d1 d2 . . . dk) s1 s2 . . . sn)
```

A statement is an action to be taken. Any expression (other than a variable) can be used as a statement, but not every statement can be used as an expression. When an expression appears in a context where a statement is expected, the expression is evaluated, but the value is discarded. A statement may have one or more labels associated with it; these are referred to in GO statements (see below) and indicate where to transfer control. Variables can not be statements because of the conflict with labels.

When evaluation of a block begins, bindings are simultaneously created for each internal parameter specified by a block declaration. These bindings remain in existence until the evaluation of the block is completed, at which time they disappear. Each binding contains a value for the variable that it binds. The nature of the binding is specified by the block declaration that creates it. After the bindings have been made, execution of the statements in the block begins. The statements are executed in turn unless the sequence of control is altered by a GO statement or by a RETURN statement. Execution of the block is terminated either by executing a RETURN statement or by executing the last statement of the block without a transfer of control.

A block declaration in SL is in the form:

```
p1 p2 p3 s1, s2, . . . , sn
```

The p_i consist of a type, a storage mode, and a transmission mode (in any order). Lexical storage and transmission by value are specified by omission; if the type is omitted, a default type is used. If all p_i are empty, the symbol DECLARE must be used. Each of the s_i is either the name of a variable or in the form:

```
v ← e
```

where e is an expression giving an initial value for the variable v . If no initial value is given, a default

value, depending on the type, is used. A block declaration causes all the specified variables to be internal parameters of the block and to have the properties specified by the p_i .

In IL, each declaration specifies the properties of one and only one variable; thus, in the translation from SL to IL, it is necessary to break up each declaration that declares more than one variable into a sequence of declarations (with appropriate factoring of properties). An IL declaration is in the form:

```
(v p1 p2 p3 p4)
```

where one of the properties is the initial value, if any.

The various types of statements and their effects may be summarized as follows:

1. *GO statement*—transfers control to the named statement.

2. *RETURN statement*—terminates evaluation of a block and determines the value of a block expression.

3. *Compound statement*—permits the insertion of a sequence of statements in a context where only a single statement is expected. A compound statement is in the form of a block with no declarations.

4. *Conditional statement*—selects one of several possible statements to be executed on the basis of the truth or falsity of a sequence of Boolean expressions.

5. *Simple expression*—causes the evaluation of the expression; the value is discarded.

6. *FOR statement*—causes an iteration to be performed for a sequence of values of a named variable.

7. *TRY statement*—causes control to be returned to itself if an exit condition is detected during the execution of a statement within the TRY statement.

8. *Block statement*—like a compound statement, except that internal parameters may be declared in the same manner as in a block expression.

9. *CASE statement*—selects one of several possible statements to be executed on the basis of the value of an integer-valued expression.

10. *Empty statement*—can be used to place a label; contains nothing and makes no action.

The FOR statement has some unusual features that merit further discussion. The statement:

```
FOR v IN x DO s
```

causes the statement s to be executed for each element of the list x , with v assuming the successive

elements as its value in each execution of *s*. If ON is used instead of IN, *v* first assumes as values the entire list *x*, then its successive terminal segments CDR *x*, CDDR *x*, etc., until the list *x* is exhausted. The clause:

UNLESS *b*

may be inserted as part of a FOR statement to inhibit execution of the statement *s* whenever the Boolean expression *b* is TRUE. The UNTIL clause of ALGOL, used in conjunction with STEP, is replaced by a relational operator and an expression; iteration continues until the variable of iteration no longer satisfies the specified relation. This approach avoids the need to recompute the sign of the increment for each iteration.

Functions

A function definition is a specification of a computational procedure; the procedure itself is a function. A function definition in SL is in the form:

t FUNCTION *n* (*x*₁, *x*₂, . . . , *x*_{*n*}); *d*₁, . . . *d*_{*k*}; *e*

where *t* is the type of the value of the function, *n* is the name of the function, the *x*_{*i*} are dummy variables that stand for its arguments, the *d*_{*i*} are declarations governing the arguments, and *e* is an expression whose value is the value of the function.

The corresponding form in IL is:

(FUNCTION (*n* *t*) (*d*₁ *d*₂ . . . *d*_{*k*}) *e*)

where a declaration is given for each argument. Thus the declarations not only give the properties of the arguments but also name them. If the value type of the function is omitted, then the name *n* can be written without parentheses and the default type will be used.

The argument parameters are used to denote the values of the actual arguments within the body of the function definition. The body of the function definition *e* is the expression that defines the value of the function. The argument declarations specify the type, transmission mode, and storage mode of the arguments.

Functional Data. A function may be used in either of two ways: as an operator or as a datum. We have already seen how functions can be used as form operators. An example of the use of a function as a datum would be the input to a numerical integration

routine; the input is the function to be integrated, and the output is the integrand. An example oriented more closely to symbolic data processing would be the use of the LISP function MAPCAR, whose arguments are a list to be transformed and a transformation function. The output of MAPCAR is the transformed list. Thus

MAPCAR ('(2 5 4 9), FUNCTION ADDER
(J); INTEGER J; J+2)

would evaluate to the list:

(4 7 6 11)

Since a function is itself a datum, it can be used in any context where a datum is expected. Thus, functions can themselves be used as arguments of other functions, and functions can be values of variables. A function can be designated by its definition, by its name, or by a variable having the function as its value.

There are two contexts in which a function may be referenced—as a datum, as we have just said, and as a form operator. When a function is used as a form operator, it must be designated either by a functional variable (i.e., a variable whose values are functions) or by a function name. The effect of using a function definition as a form operator can be achieved by assigning the function definition to a functional variable (which is legitimate, since the function definition then appears in a data context) and then by using the functional variable as the form operator.

Functions of an Indefinite Number of Arguments. It is possible to define functions that expect an indefinite number of arguments. In defining such a function, there is no way to enumerate the names of the arguments; therefore an argument vector, i.e., a one-dimensional array having a single variable name *v*, designates the set of arguments. The length of the vector is specified by a second variable *k*. In the argument list, the argument vector (which must be the first argument) is designated by writing *v*(*k*) in SL and (*v* INDEF *k*) in IL. When the function is entered, the value of *v* is the vector of arguments, and the value of *k* is the length of this vector. The different elements of the argument vector can then be referred to within the body of the definition by subscripted occurrences of *v*.

For example, the function SUMSQUARE might be written to take the sum of the squares of its arguments. We would then define it in SL as follows:

```

REAL FUNCTION SUMSQUARE(X(I));
  BEGIN INTEGER J; REAL Y;
    FOR J←1 STEP 1 UNTIL > I DO
      Y←Y + X(J)↑2;
    RETURN Y
  END

```

Here X is the argument-vector parameter and I is its length. The corresponding IL definition is:

```

(FUNCTION (SUMSQUARE REAL) ((X
  INDEF I))
  (BLOCK ((J INTEGER) (Y REAL))
    (FOR J (STEP 1 1 GR I)
      (SET Y(PLUS Y (EXPT (X J)2))))
    (RETURN Y)))

```

An actual use of SUMSQUARE might look like:

```
SUMSQUARE (2, 7, 4)
```

in SL, and:

```
(SUMSQUARE 2 7 4)
```

in IL.

Sections

A section is a collection of declarations and definitions that operate as a unit. Dividing a large program into sections makes it possible to write different parts of the program independently without name conflicts. It also makes it possible for one user to refer to programs written by another user without name conflicts. A section is designated by its section name, which is an identifier. Each section is associated with a set of variables that designate the various entities defined within the section. At any given time there is a single active section, which is known as the current section; all other sections are external sections. A variable in a particular section, whether current or not, can be referred to by tailing (often called "qualifying") e.g., "JOE\$SAM" refers to the variable JOE in section SAM.

The section mechanism permits parts of LISP 2 programs to be written and checked out independently. At merge time, attention need be paid only to variables used for names of common functions and communication variables. Since the system programs are in a special section, the user need not worry about name conflicts; at the same time, the system programs are accessible to the user through the tailing mechanism. Thus the user can, if he chooses, treat the system programs as an extension of his own program rather than as a black box.

Supervisor Level Operations

LISP 2 is controlled by a supervisor program that is itself named LISP and that can be called as a function. When the user starts up the LISP system, the supervisor is called immediately. The supervisor accepts commands to perform various operations. The actions taken by the supervisor in response to these commands are known as top-level operations. The following top-level operations are possible:

1. Evaluate an expression
2. Establish a current section with given name and default type
3. Create a fluid or own variable of specified type and transmission mode
4. Define a function
5. Define a dummy function (used to establish type information in certain cases)
6. Define a macro
7. Define an instruction sequence to be used in compilation
8. Define an assembly-language program
9. Declare a variable to be synonymous with another variable.

The user can specify the input and output devices to be used; the on-line typewriter is taken as the default case. After each operation the system sends any necessary output to the output device and proceeds to the next operation.

Input/Output. One of the primary design aims in LISP 2 I/O has been the maintenance of as much machine independence as possible. This is accomplished by distinguishing user interfaces from system interfaces and insulating the user from the system interfaces. This effect is achieved by creating machine-independent data aggregates called "files," and permitting the user to operate with files by means of LISP 2 functions.

To the user, a file is a source or sink for information, which is filled on output and emptied on input. A file itself is both device- and direction-independent. The relationship of a file to an external device is determined by the user at run time, when he specifies whether the file is to be an input file, an output file, or both.

To the system, a file consists of a sequence of records, represented internally as an array of type OCTAL if the file is binary, and as a string if the file

is composed of characters. (ASCII 8-bit characters are used internally throughout LISP 2.) To reduce buffer storage overhead, only one record for a given file can be in main memory at a time. String records are further structured into lines. The number of characters per line and lines per record may be specified by the user, but must be consistent with the conventions used by the external monitor system.

When a record in a file is moved from an external device into core, it is transformed into a LISP 2 string. The transformation may involve character code conversions and insertion or deletion of control characters. The transformation is governed by a collection of control words associated with the file. During output, this transformation, known as "string post-processing," is reversed.

File Activation and Deactivation. A file may be either active or inactive; an active file, in turn, may be either selected or deselected. No record is kept within LISP 2 of inactive files; however, many files may be active concurrently.

A file is activated by evaluating the function OPEN which establishes all necessary communication linkages between LISP 2 and the monitor. The file is named by an identifier that is its referent throughout its active life. The user further specifies the desired file description at this time. This description is given only once and consists of a list of file properties desired by the user, such as the unit (tape, disc, teletype, CRT, etc.), form (binary, ASCII, BCD, etc.), format (line and record sizes), and various protection and identification parameters.

Deactivation of a file is achieved by evaluating the function SHUT. SHUT breaks all the communication linkages and deletes all internal structures such as arrays, strings, and variables that were dynamically established by OPEN. The user may specify the disposition of the file, e.g., the saving of the tape or the insertion of the file in disc inventory. The external monitor is informed of such actions by LISP 2.

File Selection. At any given time, exactly one file is selected for input and one for output; all other active files are deselected. The LISP 2 reading functions all operate on the currently selected input file; the printing functions all operate on the currently selected output file. The functions INPUT and OUTPUT are used for selecting the input file and the output file, respectively.

When a new file is selected, the record, line, and column controls for the deselected (replaced) file are preserved, and the new file record, line, and column

controls are reestablished. Once a file is selected, all I/O primitives act only on that file. Thus it is possible to write a LISP 2 program that is independent of form, format, and device by supplying the name of the file as an argument of the program at run time. This scheme allows a LISP program to be debugged with files generated on-line and subsequently run with bulk data from tape or disc files simply by changing the selected file.

Other I/O Functions. A variety of I/O functions are available for reading and writing binary and symbolic data. There are character-level primitives that permit testing, printing, reading, and transforming characters. Other functions allow reading and printing at the token and S-expression levels. Character mappings permit LISP 2 to communicate with restricted character-set devices.

Examples

An example is now given of a complete SL program. The example includes not only the program itself but also the control actions necessary to test it:

```

SYMBOL SECTION EXAMPLES, LISP;
% LCS FINDS THE LONGEST COMMON SEG-
% MENT OF TWO LISTS L1 AND L2
FUNCTION LCS(L1,L2); SYMBOL L1, L2;
  BEGIN SYMBOL X, Y, BEST ← NIL; INTE-
  GER K←0, N, LX←LENGTH(L1);
  FOR X ON L1 WHILE LX > K DO
  BEGIN INTEGER LY ← LENGTH (L2);
  FOR Y ON L2 WHILE LY > K DO
  BEGIN N ← COMSEGL (X,Y);
  IF N ≤K THEN GO A;
  K ← N;
  BEST ← COMSEG (X,Y);
  A: LY ← LY - 1
  END;
  LX ← LX - 1
  END;
  RETURN BEST;
END;
% COMSEGL FINDS THE LENGTH OF THE
% LONGEST INITIAL COMMON SEGMENT
% OF
% TWO LISTS X AND Y.
INTEGER FUNCTION COMSEGL (X,Y);
  IF NULL X OR NULL Y OR CAR X /=
  CAR Y
  THEN 0 ELSE COMSEGL (CDR X, CDR
  Y) + 1;

```

```

% COMSEG FINDS THE LONGEST INITIAL
% COMMON SEGMENT OF TWO LISTS X
% AND Y
  SYMBOL FUNCTION COMSEG (X, Y);
  IF NULL X OR NULL Y OR CAR X /=
    CAR Y
    THEN NIL ELSE CAR X . COMSEG(CDR
      X, CDR Y);
% LENGTH COMPUTES THE LENGTH OF L
% INTEGER FUNCTION LENGTH (L); SYM-
% BOL L;
  BEGIN INTEGER K ← 0; SYMBOL L1;
  FOR L1 IN L DO K ← K+1;
  RETURN K;
END;
LCS ('(A B C B C D E), '(B C D A B C D E F));
STOP
machine: (B C D E)
    
```

This example illustrates the use of list processing capabilities combined with integer arithmetic and iteration. The operator “<=” means “less than or equal to,” and the operator “/=” means “not equal to.” The LISP operators CAR, CDR, and NULL are all used as prefix operators without parentheses. The dot in the third line of COMSEG is an infix operator

that stands for the LISP function CONS. The statement “FOR X ON L1” causes iteration to take place on the successive terminal segments of L1. Thus, if L1 is the list (A B C D), then iteration takes place successively on (A B C D), (B C D), (C D), and (D). The function LENGTH, defined here, is available as a system function and is redefined only as an illustration.

THE PROGRAMMING SYSTEM

System Overview

A diagram of the LISP 2 system which shows the relationship among its different components is shown in Fig. 2. Information enters the system via the I/O package in either SL or IL. The I/O package transforms the input into a stream of characters—the input to the finite state machine—which in turn generates a stream of tokens. Among other things, the finite state machine performs the task of linking up a newly received identifier with a previous copy of the same identifier. The token stream produced by the finite state machine is routed by the supervisor to either the syntax translator or to a reading program for IL, depending on whether SL or IL is expected. In either case, the result is an ex-

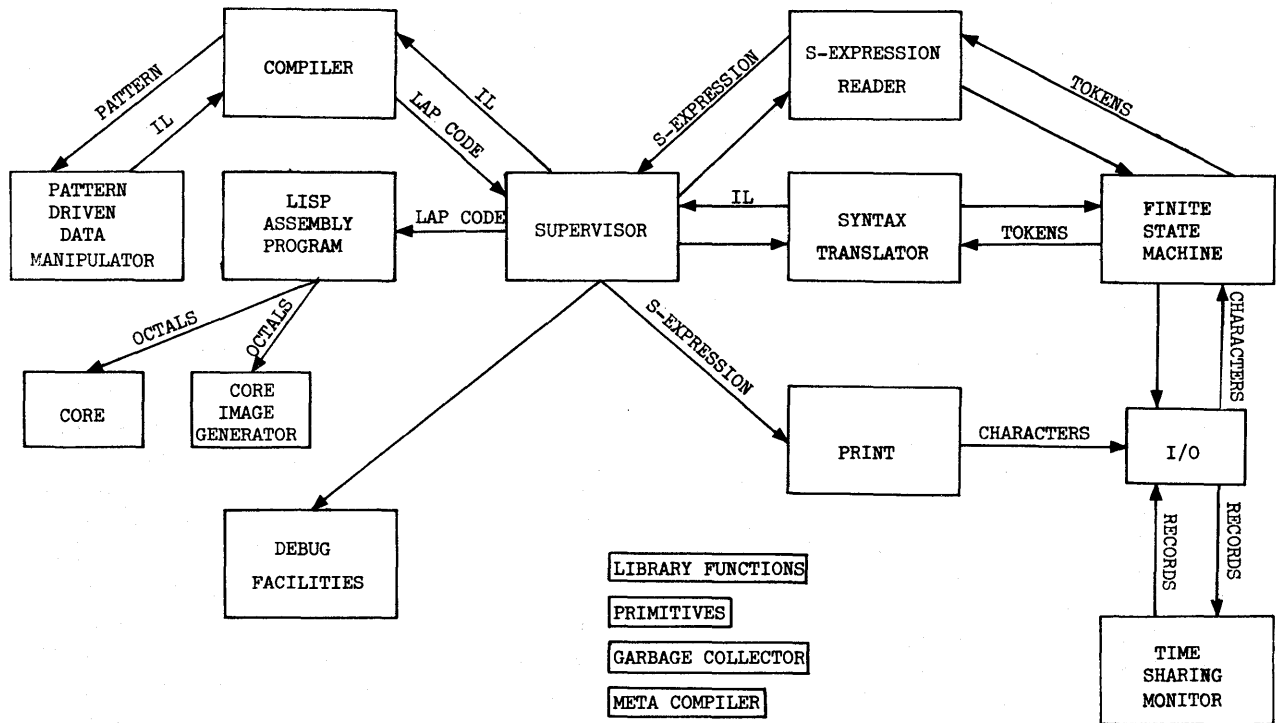


Figure 2. System components and information flow paths (unlabeled connections designate control paths).

pression in IL. The supervisor determines when compilation is to take place, and also handles processing requests.

The syntax translator takes a stream of SL tokens and transforms it into an IL expression. This expression can be returned as output, passed to the compiler, or both. The choice is made by the supervisor under the control of the user. The syntax translator consists of parsing and generating programs that are compiled from a set of syntax equations. These syntax equations define SL in terms of IL.

The compiler, which is the most complex component of the system, converts IL into input for LAP, the LISP Assembly Program, or for the core image generator. Both LAP and the core image generator accept input in assembly language (AL). If LAP is being used, then the result of assembly is a relocatable segment of code stored in an area of the machine reserved for binary program. If the core image generator is being used, then the result is a string of pairs of binary numbers, each consisting of a core location and the contents of that location, stored on a magnetic tape or other external medium. The core image generator is only used when a new system is being created.

The META compiler, the garbage collector, and the primitives are all implicitly involved in the operation of the system. The META compiler is a library program that generates a syntax translator from a set of syntax equations. The garbage collector is the program that collects dead storage when available storage has been exhausted. The primitives are the basic library functions in terms of which the entire system is written.

Memory Management

Most of the concepts of memory management used in LISP 1.5 are also used in LISP 2. Memory management in LISP 2 is based on several considerations:

1. LISP 2 data structures may vary in size by orders of magnitude at run time, and storage for such data structures must be allocated automatically.
2. Since recursion is permitted, successive generations of data structures must be retained simultaneously.
3. Programs and data structures that are no longer needed must be purged without explicit action on the part of the user.

4. Numerical data must be stored in such a way as to permit efficient numerical calculations.

LISP 2 data structures may be either variable or fixed in size. The variable data structures are arrays, strings, and symbolic expressions. Although an array, once established, does not change in size, the size of an array is frequently not known until the occasion arises to create it. In the case of list structures, the situation is even more complex; a list structure may be modified in such a way as to increase or decrease its size.

Arguments of functions and internal parameters of blocks are stored on a pushdown stack. Since all temporary storage belonging to LISP 2 functions is recorded on the pushdown stack, which is maintained by the LISP 2 system, recursion is permitted with no special user provisions. Unlike LISP 1.5, LISP 2 stores numbers directly on the pushdown stack as single cells. Therefore, it is possible to perform arithmetic without the loss of efficiency that would arise from packing and unpacking numbers referenced indirectly. Symbolic expressions, strings, and arrays, however, are accessed by means of pointers stored in the stack. The data structures thus pointed to are discarded when the function creating them has completed its execution; however, they do not disappear, but remain as garbage until the next garbage collection, the description of which follows.

In LISP 2, data structures are grouped according to their storage characteristics and a storage area is set aside for each group. The groups are:

1. Elementary symbolic entities (symbolic constants, function and variable names, etc.)
2. Compiled programs
3. List structures
4. Arrays and strings

In addition, a storage area is set aside for the pushdown stack. These storage areas are arranged in pairs, where one member of the pair grows from the bottom up and the other grows from the top down. Data storage is obtained by taking storage space from the appropriate area until that area is exhausted (which occurs when its boundary meets the boundary of the area that is paired with it). At this point, the garbage collector is invoked. Garbage collection erases all inaccessible data structures and reclaims the emptied space for new structures. For instance, if a LISP 2 function has been redefined, the program

corresponding to its old definition is inaccessible and thus is erased. During garbage collection, the different areas are compacted, relocating code and/or data structures, if necessary, so as to eliminate the gaps left by erased structures.

The different kinds of structures are stored in different areas because their requirements in terms of garbage collection are different. For instance, the elementary symbolic entities cannot be moved, but other kinds of data can be moved. Similarly, list structures consist of independent nodes, while arrays consist of blocks of different sizes.

The Syntax Translator and the META Compiler

The translation from SL to IL is performed by a syntax translator that was generated by the META compiler. The META compiler is based upon a program developed by Special Interest Group for Programming Languages of the Los Angeles Chapter of ACM.⁸ The META compiler takes as input a specification of the syntax of SL, together with instructions on how each syntactic entity is to be transformed to IL. It produces an IL program that actually carries out the translation from SL to IL. The description of the syntax of SL is given in an extended version of Backus-Naur Form.⁴

The META compiler produces top-to-bottom compilers with a controlled backup feature and an interface with the finite state machine (see below). Both the controlled backup and the finite state machine are efficiency features. The controlled backup allows the designer of a language to specify in the syntax equations when the state of the machine must be saved because two or more parsings start with the same construction.

As it is possible to regenerate the syntax translator with new syntax equations at any time, the syntax and semantics of SL are not, in principle, rigidly fixed. In practice, variants on the syntax translator will be used in order to translate other languages into LISP 2 IL. These other languages, unlike SL, will normally not be semantically equivalent to IL.

Finite State Machine

The finite state machine (FSM) is a token-parsing program used by the syntax translator and the S-expression reader. Reading LISP 2 entities is expensive, not only in the original creation of the internal structures, but also in the time spent in

garbage collecting when the structures are discarded. Consequently, it is desirable to avoid backup at the character level and its resulting re-creation of duplicate structures. Since backup must be used by the syntax translator, the FSM was imposed between it and the character stream to eliminate reprocessing of tokens. Having the bottom-to-top FSM interface with the top-to-bottom syntax translator eliminates a large portion of the overhead associated with reading in the LISP 2 system. The S-expression reader does not require backup, but since the FSM existed, it was convenient to use tokens for building S-expressions also.

The FSM behaves like a Turing machine. It moves from state to state as it reads characters; when a terminal state is reached, it "prints" a character from its output alphabet (tokens) and sets its state to the initial one. Parsing and manufacture of structures are done simultaneously as characters are recognized. No reprocessing of the parsed characters is ever necessary, since in a terminal state the token is already complete (except for a final action, such as combining the parts of a real number).

The LISP 2 Compiler

The LISP 2 compiler is a large, one-pass, optimizing translator whose input is a function definition in IL and whose output is an assembly-language list of instructions suitable for input to LAP. Most of the compiler is independent of the target machine, since the compilation concepts themselves are machine-independent. The declarations of all fluid variables appearing within the function are written into the output listing, since these must agree with fluid variable declarations made elsewhere. Checks are made for both format and semantic errors during compilation. The compiler consists of three major sections: the analyzer, the optimizer, and the user control functions.

Analyzer. The top-level control of the compiler resides in the analyzer, which operates recursively. Each item to be compiled is passed to the analyzer either directly or indirectly. If the item is a variable, an appropriate declaration is found and code for retrieving the variable is generated; otherwise the code for a function call is generated, a macro expansion is performed and the result compiled, or linkage to an appropriate code generator is made. A pattern-matching function has been implemented for use in the LISP 2 compiler. The patterns are written in a modified form of Backus-Naur Form (not the same

as the one used in the syntax translator). The patterns are matched to an S-expression and the value of the match is either TRUE or FALSE. The pattern-matching function checks for syntactic correctness and distinguishes among different forms at the same time.

Optimizer. Optimization of the code produced by the LISP 2 compiler is handled by many groups of routines, each responsible for certain actions. The communicative mechanisms between these various parts and the rest of the compiler will be described in some detail below.

The movers, a highly machine-dependent set of functions, produce code that alters the state of a compilation in a specified way, such as moving an object to an accumulator or converting a datum to a specific type. Embodied in the movers is a predicate capability that answers the question, "Is this move possible under these conditions (say, one machine instruction)?" The movers are used to build all address and modifier fields of generated instructions. Associated with the movers is a post-processor that rewrites the output code after the main compiler has produced it. Redundant load-store sequences and some unnecessary branches are removed from the listing. Also, certain groups of instructions are rewritten to make use of machine-specific instructions.

The arithmetic optimization package handles code generation for addition and multiplication. The algorithm that is used is a standard one, namely, first sorting the arguments by type and then by priority sequence within a particular type. The sequence depends on whether the arguments are memory or accumulator references. A single set of functions handles both multiplication and addition, with the aid of several functional arguments.

A second kind of optimization has to do with the elimination of unnecessary transfer instructions. This task is accomplished through the analysis of confluence points, i.e., places in the program at which several paths of control converge. For instance, consider the conditional expression:

$$(\text{IF } p_1 \ e_1 \ p_2 \ e_2 \ \dots \ p_n \ e_n)$$

The appearance of this conditional expression establishes a confluence point at the end of the compiled code that represents it. After the execution of any of the e_i , control goes to this confluence point. Moreover, the confluence point is hereditary for each of the e_i , i.e., if one of the e_i is a conditional expression, then its confluence point is the same as that of

the entire expression. Analogous considerations hold for conditional statements. Confluence points are also hereditary with respect to RETURN statements of blocks, i.e., the confluence point of a RETURN statement is the same as that of the block in which it appears.

When an expression is compiled, the characteristics of the value that is produced must be specified. These characteristics include type, whether it is in a special register or in an ordinary memory cell, its address modifier (direct or indirect), which registers it may be left in, whether the actual value is needed or whether the negative or reciprocal of the value is so described, etc. These characteristics are remembered by a set of state variables, which are bound for each call to the analyzer. As a statement or expression is compiled, a listing is generated and the state variables set to reflect the state of the compilation. The compiler is passive in the sense that a compilation produces only the minimum amount of code necessary to allow the result to be described by the state variables.

User Control Facilities. The user can give the compiler explicit instructions to aid in the compilation process. As in LISP 1.5, macros are an integral part of the language. Many of the facilities of the language, e.g., FOR statements, are implemented by means of system macros. When a FOR statement (in IL form) is encountered during compilation, it appears as an operational form whose operator is FOR. The compiler tests each form operator to see if a macro is defined for it. In the case of FOR, there is such a macro. The macro is invoked with the FOR statement (in the form of an S-expression) as input. The output is a block containing an equivalent iterative loop. This block is then compiled in place of the FOR statement. Macros may also be defined by the user, and no distinction is made between system macros and user macros.

Certain machine-dependent operators are particularly useful as primitives in compilation. CORE is an operator that acts like an array whose content is all of the machine memory. Therefore CORE(x) is the content of location x. BIT is an operator that specifies a certain contiguous portion of a word. There are also several operators that permit an expression to be forced to a certain type or permit a datum of one type to be used as though it were of another type. Although such mechanisms exist in most compilers, LISP 2 has made these items available through the language.

The LISP 2 Assembly Program

The LISP 2 Assembly Program, LAP, is a program that generates a code segment from a list of symbolic instructions and labels. LAP also allocates storage for variables on the pushdown stack, and insures that references to fluid and own variables are consistent among different compiled functions. LAP does more than most assemblers, in that it handles all aspects of pushdown stack mechanics; consequently, references to variables are made by naming the variable in the appropriate field of any instruction that references it. Thus, the pushdown stack need never be referenced explicitly.

LAP includes a number of system macros specifically designed for LISP 2 programming. The prologue and epilogue of a function are generated by BEGIN and RETURN respectively; CALL is used to generate a call to a LISP 2 function in the standard format. Storage allocation on the pushdown stack is performed by the BLOCK, DECLARE, and END macros; FLBIND creates any necessary bindings for fluid variables. LAP does not have a generalized macro facility; any effect that could be achieved by such a facility, however, can also be achieved by preprocessing.

The address field of an instruction may be used to allocate, refer to, or release temporary storage on the pushdown stack. The address fields TOP. and POP. are normally used with instructions of the "load" type. Both TOP. and POP. refer to the most recently allocated pushdown cell, but POP. has the additional effect of releasing that cell. PUSHA. and PUSHP. both cause a new pushdown cell to be allocated, and refer to that cell; PUSHA. and PUSHP. are normally used in instructions of the "store" type. PUSHA. is used for absolute quantities and PUSHP. for symbolic quantities, so that a map of the pushdown stack can be maintained.

To illustrate the use of assembly language, as well as the output code produced by the compiler, we give the Q32 assembly language version of the program RANDOM presented as an example earlier in the paper:

```
(LAP (FUNCTION (RANDOM REAL)
  ((A REAL) (B REAL))
  (STF TOP.)
  (BEGIN)
  (LDA Y)
  (MUL 3125 (L567.7 R S))
  (STB Y)
```

```
(ARGS)
(LDA Y)
(STF PUSHA.)
(LDA (NUMBER 67108864) S)
(CALL (REMAINDER . LISP))
(STF Y)
(LDC A)
(FAD B)
(STF PUSHA.)
(LDA Y)
(FLT (ENTRY B48.))
(FDV (NUMBER 6.7108864000E-7))
(FMP POP.) (FAD A) GO9017 (END) (RE-
TURN))
(((REMAINDER . LISP) FUNCTION (FUNC-
TIONAL INTEGER INTEGER INTEGER)
NIL) (Y OWN INTEGER NIL)) USER)
```

ACKNOWLEDGMENTS

LISP 2 is being developed jointly by Information International, Inc., and System Development Corporation, with contractual support from the Advanced Research Projects Agency of the Department of Defense. Personnel actively participating in this program include:

Dr. Paul W. Abrahams (III)
 Mr. Jeffrey A. Barnett (SDC)
 Mr. Erwin Book (SDC)
 Mrs. Donna Firth (SDC)
 Mr. Lowell Hawkinson (III)
 Dr. Stanley L. Kameny (SDC)
 Mr. Michael I. Levin (III)
 Mr. Robert A. Saunders (III)
 Mr. Clark Weissman (SDC)

In addition, we wish to acknowledge the voluntary support and contributions received from Professor Marvin Minsky and his associates at MIT, Professor John McCarthy and his associates at Stanford University, Dr. Daniel G. Bobrow of Bolt, Beranek and Newman, and many others.

REFERENCES

1. M. Levin, "LISP 2 Primer," SDC Document TM-2710/101/00 (July 15, 1966),
2. T. Abrahams, "LISP 2 Reference Manual," SDC document in preparation.

3. M. I. Levin, *LISP 1.5 Programmers Manual*, MIT Press, Cambridge, Mass., 1962.

4. "Revised Report on the Algorithmic Language ALGOL 60," *Comm. ACM*, vol. 6, no. 1, pp. 1-17 (1963).

5. V. Yngve, *COMIT Reference Manual*, MIT Press, Cambridge, Mass., 1962.

6. D. G. Bobrow, "METEOR, a LISP Interpreter

for String Transformation," in "The Programming Language LISP," Information International, Inc., Cambridge, Mass, 1964, pp. 161-90.

7. "ALGOL algorithm #266," *Comm. ACM*, vol. 8, no. 10, p. 605 (1965).

8. D. V. Schorre, "META II, a syntax-directed compiler writing language," *Proc. ACM*, p. D1.3-1 (1964).

APL—A LANGUAGE FOR ASSOCIATIVE DATA HANDLING IN PL/I

George G. Dodd

Research Laboratories, General Motors Corporation, Warren, Michigan

INTRODUCTION

Engineering design and computer-aided problem solving occur in an atmosphere in which the relationships between the data elements are utilized. For example, information retrieval,^{1,2} computer-aided drawing,³ electrical network design,⁴ and engineering design⁵ systems are among those whose operation depends on efficient data manipulation and association techniques.

Problem solving with a computer should give a user an opportunity to continually change and restructure his data. In effect, he should be able to reach into his data structure and ask questions about the data: What pieces of data is this one related to? What is affected if the numeric value attached to the data element is altered? Likewise, he should be able to specify operations to be applied to a group of elements: Collect these things together! To that collection of data elements apply algorithm XXX! Create a new element and put it into a collection!

Algebraic programming languages permit elaborate mathematical expressions to be stated in a clear and direct manner. The user who is uninterested in assembly language techniques can state a procedure for arriving at a solution to a problem, in a high-level language.

The solution of many classes of problems, however, requires data handling and here the existing languages break down. The algebraic languages do

not permit the user to treat his data in the same clear and direct manner as he treats mathematical formulae. He has to become an expert in two fields—the one, his specialty and the other, data handling.

APL was conceived at the General Motors Research Laboratories to satisfy the need for convenient data association and data handling techniques in a high-level language. Standing for ASSOCIATIVE PROGRAMMING LANGUAGE, it is designed to be embedded in PL/I⁶ as an aid to the user dealing with data structures in which associations are expressed.

The language offers facilities for establishing relations between the data elements, developing and restructuring the data structure as the program proceeds, and manipulating data without programmer concern about the complicated mechanics used to establish the data relationships. In addition to a full PL/I capability, the language provides

- a. Symbolic data references.
- b. Use of English verbs and phrases in data manipulation statements.
- c. Hierarchy of constituent parts permitting orderly formulation of data relationships.
- d. N-dimensional data associations.
- e. Automatic extension of addressable memory beyond the confines of high-speed core.

Machine independent implementation is possible by writing the APL preprocessor in PL/I. However, the deferred and incomplete pointer features of PL/I have made this impossible at the moment.

APL contains many of the facilities underlying such list processors as IPL⁷, COMIT⁸, LISP⁹, SLIP¹⁰ but it gives the user a more complete data manipulation capability with less effort. Unlimited block sizes accommodate any PL/I structure and the data linking techniques offer a full class of forward, reverse, and n-dimensional associations between the data structures.

ELEMENTS OF AN ASSOCIATIVE DATA STRUCTURE

The associative data structure underlying APL can best be described in terms of the hierarchy of its elements. The structure is the framework in which the data items reside. Each structural element is tied to other elements by means of their relationship, and can have other elements tied to it. In addition, each structural element is described by data pertaining only to that member.

The basic element of an associative data structure is an ENTITY. Any identifiable type of element in a data structure such as a job, a surface, or an automobile is treated as an entity. Entities are described by ATTRIBUTES (more precisely, the values of attributes). For example, an entity which is a Job can have the attributes Job ID, Job Title, Duration, Finish Date, and Start Date.

Many different types of entities may exist in a data structure. An engine, frame, tire, windshield are all types of entities appearing in the data structure of an automobile. In addition, many copies or instances of each entity type may abound.

Related entities may be grouped into a SET. This SET can be: (1) owned by an entity; or (2) it can be resident in data space and referenced independently of any entity. An example of the former is the collection of jobs which must be completed before a given job is started: here the collection is tied to an entity and is called a SUBSET of the entity. A set of type (2) might be the set of all jobs in the file. This set is not attached to any entity but is known to all programs using the file.

An entity is depicted as a contiguous block of addressable memory having one or more of the following:

- a. References to the subsets belonging to the entity.
- b. References to the one or more sets to which the entity may belong. These are known as associative set reference links.
- c. Data attributes associated with the entity.

The implementation technique used stores pointers in the entity block to the subsets of the entity and to the sets to which the entity belongs. In Fig. 1, this is portrayed with the X followed by an arrow indicating the subset links and the double-ended arrows indicating the associative reference links.

Entities are linked into a set by means of Roberts' ring structure.* This structure was chosen because it permits rapid movement through a set in one direction. At the same time it permits movement through a set in a backwards fashion or to the entity to which the set is tied with only a small increase in overhead. Because of these properties, random addition and deletion of entities in a set is easily accomplished.

Figure 2 shows entities {B,C,D} connected to a set belonging to A. This is expressed as $\{B,C,D\} \varepsilon A$.

An entity can have many subsets belonging to it and in turn can belong to an indefinite number of sets. The contiguous entity structure permits subsets and associative references to be found directly in the entity rather than moving along a list.

Figure 3 shows D and E belonging to a subset of B. E is also a member of a subset of C which is a member of a subset of A. These relationships can be logically expressed as $\{D,E\} \varepsilon B$, $E \varepsilon C$, and $C \varepsilon A$.

The data attributes describing the entities may assume one of two forms, direct or associative. A

* Copies of a paper describing this structure can be obtained from Dr. Lawrence Roberts, Lincoln Laboratory, Lexington, Mass.

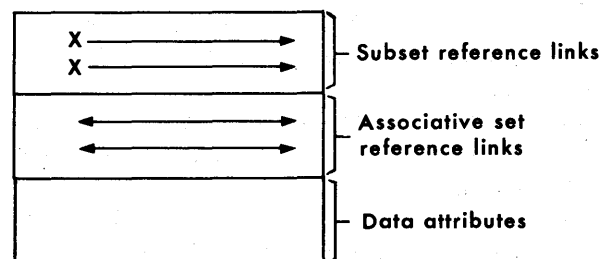


Figure 1. General entity.

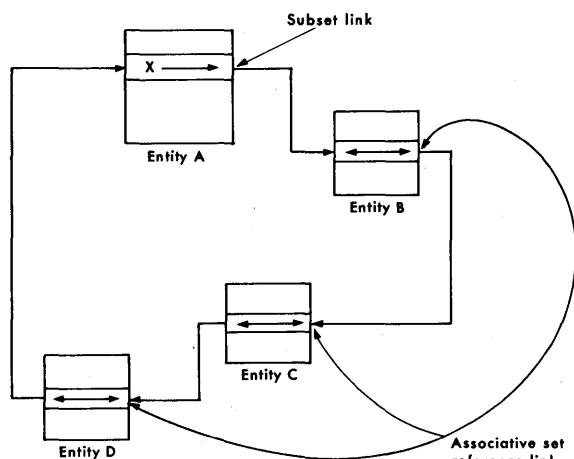


Figure 2. Set of entities.

direct attribute is a PL/I data structure residing in an entity block and addressable as part of the entity. Associative attributes are used to express relations between entities or between an entity and a set of which it is a member. Direct attributes are illustrated by the geometric coordinates which describe a point. However, the distance between that point and some other point is an attribute value associated with both points; both points as well as the name of the attribute must be known to retrieve its value.

In some cases an entity is a conditional member of a set. Here, one or more attributes are needed to fully qualify the membership. In Fig. 4 the line is a member of the boundary line set for surface 1 between the value X_1 and X_3 and a member of the boundary line set of surface 2 between the values of X_2 and X_4 . These associative attributes may be specified and retrieved within APL.

PROGRAMMER TOOLS

Now that the elements of an associative data structure have been described, let us turn our attention to how it is used. The tools for the data manipulation can best be described in terms of

- a. Data set declarations,
- b. Attribute set and entity references,
- c. Data structure manipulation statements.

Data Set Declaration

The entities, sets, and attributes making up an associative data structure are specified in a series of statements preceding the programs to be translated

and compiled. This procedure insures compatibility across jobs and eliminates repetitive declarations in the programs compiled within the same job environment. Sufficient information is provided for the opening of files or initialization need prior to the execution of the first instruction in the program.

The specification of an entity includes the entity type, i.e. an alphanumeric designation by which that type of entity will be referenced, the names of the subsets of the entity, the names of the associative sets linking the entity and the names of the direct and associative attributes of the entity.

A separate declaration of each set is also required. In these statements the procedure for the storage of entities within the set is specified. The storage procedures can be overridden by a specification from the programmer any time in his program. (Figure 8, appearing later, shows the declaration of a JOB entity having a PRED subset, associative links to the PRED and ALLJOB sets, and a number of direct attributes.)

Associative attribute declarations are prefaced with an ATTRIBUTES* identifier. A point entity having direct attributes x, y, z and an associative attribute *distance* is declared by

```

DECLARE 1 point ENTITY,
      •
      •
      •
      2 distance ATTRIBUTE* float,
      2 x ATTRIBUTE float,
      2 y ATTRIBUTE float,
      2 z ATTRIBUTE float;
    
```

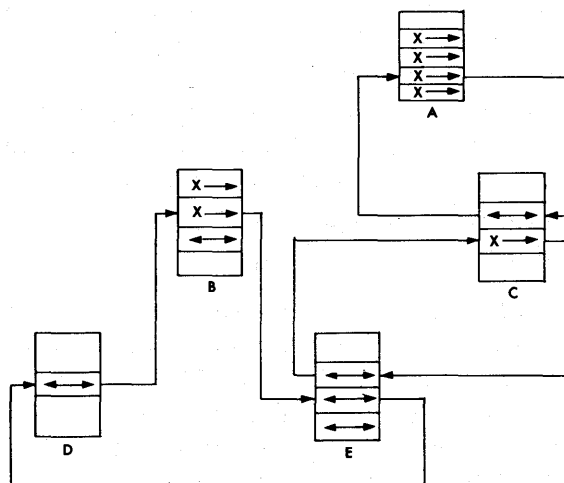


Figure 3. Multiple sets of entities.

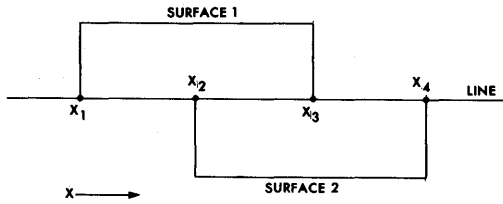


Figure 4. Example of associative attribute.

Attribute, Set, and Entity References

Many instances of each type of entity can appear in a data structure and it is often necessary to reference several of these instances simultaneously. To do this a variable of data type ENTITY has been introduced in APL. At the time an entity is created or obtained through a referencing mechanism, an entity variable is set to identify the correct instance. Figure 5 shows five entities; one being referenced by entity variable A, one by entity variable B, and three un-referenced.

Sets are referenced through the entity instances to which they belong by means of the notation

$$\langle \text{entity instance} \rangle . \langle \text{set name} \rangle$$

In the figure, the fully qualified name of the PROP subset is given by A.PROP.

The value of a direct attribute is referenced by qualifying its name with the names of the relevant sets and entities. The names are separated by a period "." in a manner similar to the period qualifier used in PL/I data structure names. Accompanying each set name is an integer specifying the ordinal number of the desired entity. For example, if A currently references an entity whose PROP subset has a third entity whose X value is desired, the latter is referenced by A.PROP(3).X (see Fig. 5). The Z value of A is obtained by A.Z.

In another example, the X's of the first three entities in the subset PROP are added together and placed in the first Y by the assignment statement

$$A.PROP(1).Y = A.PROP(1).X + A.PROP(2).X + A.PROP(3).X$$

The attribute expression is usable in more than one level. The attribute COST of entity B may be designated by A.PROP(3).Q3(1).COST. The last entities in a set may be referenced by negative integers. Thus, A.PROP(-1).X is the last X in the subset and A.PROP(-2).X is next to the last X.

Associative attributes are referenced by the format

[entity - 1, entity - 2].attribute
OR
[entity, set].attribute

depending on whether the associative attribute is between entities or between an entity and a set. If the attribute distance existed between entities E1 and E2, its value could be specified by

$$[E1, E2] . DISTANCE = 5;$$

Both direct and associative attribute references may be used in any APL or PL/I statement where a value is to be retrieved or assigned.

Data Structure Manipulation Statements

The statements provided by APL for data manipulation have a key word syntax. The statements complement existing PL/I statements and may be used anywhere within the source program. The six statements and their meaning are:

a. CREATE *entity-type* CALLED *entity-variable*;

Space for an entity of the specified type is allocated in the users file and the designated entity variable is set to reference the new entity.

b. INSERT *entity-variable* IN *set*;

The designated entity is made a member of the designated set.

c. REMOVE *entity-variable* FROM *set*;
REMOVE *entity-variable* FROM ALL;

The entity is either removed from the designated set or from all sets of which it is a member.

d. DELETE { *entity-variable* }
 { *set* } ;

The first option deletes the specified entity from the file and its space is returned to the free-space list for reallocation. The second option removes all en-

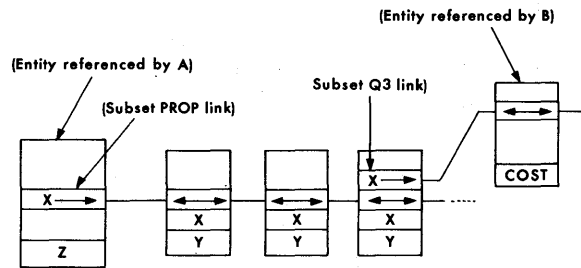


Figure 5. Attribute references.

tities from the specified set and deletes them. The delete routine is recursive in that the entities belonging to subsets of the given entity will also be deleted if they belong to no other subset.

- e. FIND *entity-variable* = *entity specification*
 [,WITH β] [,UNTIL β] [,ELSE *statement*];

In its basic form the FIND statement sets an entity variable to reference an entity whose identity is given by the clauses of the statement.

The entity specification can assume one of two forms; the specification of an entity which is a member of a set or the specification of an entity whose subset contains a designated entity. These can be illustrated by

FIND *e1* = (3) *line* \subset *e2.boundary*;

which finds the third LINE entity contained in the BOUNDARY set of E2 and

FIND *e2* = *boundary* \supset *e1*;

which finds the entity whose BOUNDARY SET contains E1. Because of the lack of characters in the PL/I 60 character set, the word IN is substituted for \subset and CONTAINS is substituted for \supset in the APL implementation.

The WITH phrase contains a Boolean expression to be applied to each entity in the set. If an entity satisfies the expression, a count is increased until the correct entity is found. The third POINT entity having an X value of 3 would be specified by

FIND *pt* = (3) *point* \subset *e1.ptset*, WITH
 $x = 3$;

The Boolean expression in the UNTIL phrase designates the condition which terminates the search. Should the search be terminated either by exhausting the set or by action of the UNTIL phrase the ELSE clause is executed. The WITH, UNTIL and ELSE phrases are all optional.

Unless otherwise specified the statement is executed on a set in the order in which entities are inserted in the set. The search can be made in a backwards direction through the set (specification of negative indices) and can be directed to start at some element in the set (by inserting \subset FROM *entity variable* \supset in the entity specification.

- f. FOR EACH *entity-variable* = *entity-specification*, [,WITH β]
 [,UNTIL β];
 END;

Simply put, this statement is a FIND statement in a DO loop. The block of statements between the FOR EACH and END are executed for each entity variable meeting the specifications.

The APL manipulation statements also augment the Boolean capabilities of PL/I with two additional tests: (1) test a set for emptiness; and (2) test an entity for membership in a designated set.

CONTROL OF EXTERNAL STORAGE

APL operates on the philosophy that external storage should merely be treated as a larger addressable memory.¹² Therefore, special PUT and GET, READ and WRITE statements are not necessary. The system operates in conjunction with a software data paging supervisor.* This supervisor intercepts all data references and supplies the program with the correct data items.

At program initialization time a buffer is established in core where, upon request, the necessary data pages are placed. Each data page contains a number of entities. The address of an entity specifies both the page within which the entity resides and the physical location of the entity within the page.

Upon intercept of a data request the supervisor examines its stock of in-memory data pages. If the correct page is present, the address of the correct entity or set in the page is returned to the program. If the data item does not reside in the core buffer, an unused or old page is transferred out and the correct page is found in the external storage and read into the paging buffer. The relocated address of the requested set or entity is then returned.

The system can also operate in a non-paging mode. Here, addresses of all data items are absolute and no paging or pointer interception occurs. Limitations of the non-paging mode are the confines of core memory for a work space and the external file manipulations required if the user wishes to save his data set.

Operation in a time sharing and paging environment is also being anticipated. At this time the paging supervisor will be lifted out and replaced by the hardware paging and segmentation functions of the system.

* The software paging mechanism forces limitations on the entity size for efficient processing. The entity size must be less than the page size. This restriction will be removed when hardware paging and segmentation is implemented.

Example

Project scheduling programs offer some interesting problems which can be readily solved by programming in APL.

Figure 6 is a simplified project schedule. This schedule shows the activities which must occur in moving the project from the MASTER PLAN to the end of TEST PRODUCT. The activities and their titles are represented by the solid lines in the figure. The dotted lines show the precedence relationship between the activities in the project. For example, MASTER PLAN has no predecessors and can begin immediately. The activity PRODUCE E has the immediate predecessors PLAN B and PLAN C and cannot be started until the predecessor jobs have been completed. The predecessors, therefore, determine the ordering of the activities.

Each activity also has a duration. This is a measure of the time required to complete the task once it has been started. In the example, MASTER PLAN, PLAN B and PLAN C have respectively 4, 7 and 13 days duration. PRODUCE E has 13 days duration but cannot be started until both PLAN B and PLAN C have been completed. This pushes its start date back to the fourteenth day.

An APL program will be written to read in descriptions of activities, establish the required data structure and compute the start and finish dates for each activity.

The input to the program is information about each job (Figure 7). This consists of an ID number and title by which the job will be identified, the duration of the job, and a listing of the immediate predecessors of the job.

The data structure used in this program has one type of entity called JOB. Each JOB entity has a subset PRED which is the collection of all immediate predecessors to the JOB. A JOB may be a predecessor of another JOB; if so the association is

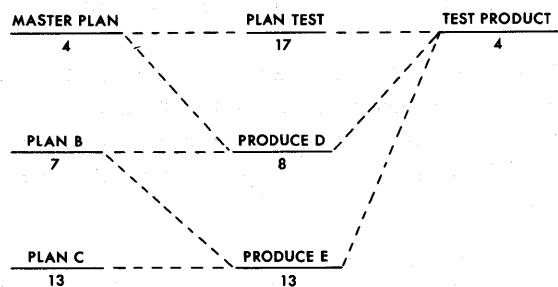


Figure 6. Project schedule.

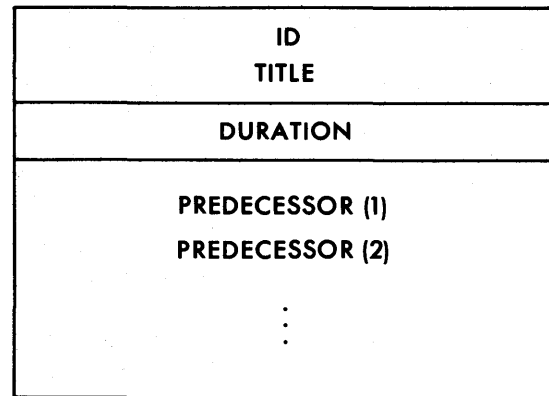


Figure 7. Job information form.

through the PRED associative set member link. These links are internally expanded so that JOB can be a predecessor of any number of other JOB entities. All JOBS are collected together in the set ALLJOB by means of a second associative reference link.

Each entity has the attributes duration and title given in the job information form. In addition, it has a start and finish date which will be calculated from the location of the entity in the precedence chart.

Figure 8 is the APL declaration of a JOB entity, the entity attributes and the sets relating the entities as outlined above.

The specification further outlines that PRED sets are a first-in-first-out assignment (FIFO) and that the ALLJOB set is ordered on the ID of its constituent entities.

```

/*DECLARE THE ENTITY CALLED JOB*/
DECLARE 1 job ENTITY,
        2 pred SET fifo,
        2 pred MEMBER,
        2 alljob MEMBER,
        2 sw BIT (1),
        2 title CHAR (20),
        2 duration FIXED,
        2 start FIXED,
        2 finish FIXED,
        2 id FIXED;
12
13
/*DECLARE THE ALLJOB SET*/
DECLARE alljob SET ORDERED INCR ON id;
14
15
  
```

Figure 8.

Figure 9 illustrates a part of the data structure to be constructed by an APL program operating on the declarations given in Fig. 8. In the structure JOB C is expanded to show the attribute and set relationships. JOBS A and B are members of the

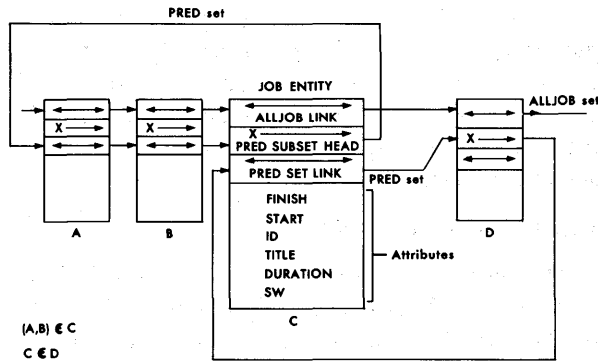


Figure 9. Data structure for scheduling program.

PREdecessor set of C and JOB C is a member of the PRED set of D. In addition C has the attributes shown.

Figure 10 shows the APL program. This program, plus the declaration statements given in Fig. 8, are what are given to the APL processor.

The program is broken into several parts:

- Statements 2-4 —declarations.
- Statement 5 —create a file known as SCHJOB. All future CREATES are assumed to be in this file.
- Statement 6 —create a subset link in the file for ALLJOB.
- Statements 7-10 —read in the ID of a new JOB and create the JOB if this has not already been done.
- Statements 11-15 —place the predecessors of the new JOB in its PRED set. If a predecessor has not been defined, create a JOB whose duration, title and predecessors are to be supplied later.
- Statements 16-23 —internal procedure that creates a JOB entity, sets its ID and SW. SW is a switch used in statements 25-28 to establish the finish dates of the activities.
- Statement 24 —read in start DATE.
- Statements 25-28 —find a JOB whose predecessors have been resolved or which has no predecessors.
- Statements 29-36 —assign the JOB a start date.

Statements 37-43 —repeat for all JOBS; when done check for unassigned JOBS (these have cycles caused by inconsistent data) and print a comment.

```

SCHEDULE:PROCEDURE OPTIONS (MAIN);      1
  DECLARE (y,z) ENTITY;                  2
    (p,date) FIXED;                      3
    test BIT(1);                          4
  CALL CRFILE ('schjob');                 5
  CREATE alljob;                          6
  READ:GET LIST (ident);                  7
  IF ident = (-2) THEN GO TO LINK; /* -2 SIG- 8
    NALS END OF INPUT*/
  FIND z = (1) job IN alljob, WITH z.id = ident,  9
    ELSE CALL DEFINE (ident,z);
  GET LIST (z.title,z.duration);         10
  MORE:GET LIST (p);                      11
  IF p = (-1) THEN GO TO READ; /* -1     12
    MARKS END OF PREDECESSOR LIST*/
  FIND y = (1) job IN alljob, WITH y.id = p,   13
    ELSE CALL DEFINE(p,y);
  INSERT y IN z.pred;                     14
  GO TO MORE;                             15
  DEFINE:PROCEDURE (arg1,arg2);           16
  DECLARE arg1 FIXED, arg2 ENTITY;       17
  CREATE job CALLED arg2;                 18
  arg2.id = arg1;                         19
  arg2.sw = '0'b;                         20
  INSERT arg2 IN alljob;                  21
  RETURN;                                 22
  END DEFINE;                             23
  LINK:GET LIST (date);                   24
  AGAIN:test = '0'b;                      25
  END2:FOR EACH z = job IN alljob, WITH    26
    z.sw = '0'b;
    FIND y = (1)job IN z.pred, WITH y.sw = '0'b,  27
      ELSE GO TO GO1;
    GO TO END3;                           28
  GO1:z.sw = '1'b;                        29
  IF z.pred = EMPTY THEN                  30
    z.start = date;                       31
  ELSE DO                                  32
    z.start = MAXVAL(finish,z.pred) + 1;   33
    z.finish = z.start + z.duration;      34
    test = '1'b;                          35
  END3:END END2;                          36
  IF test THEN GO TO AGAIN;               37
  FIND z = (1) job IN alljob, WITH z.sw = '0'b, ELSE 38
    DO; PUT LIST ('satisfactory run');    39
    RETURN;                                40
  END;                                     41
  PUT LIST ('incorrect data');            42
  END SCHEDULE;                           43
    
```

Figure 10.

Observe that some of the statements, such as CREATE and FIND, require the variable Y or Z to be the representative of each entity in the set. This is an entity variable described earlier and its declaration is in Statement 2.

At times it is convenient to designate an entity as being the current entity for purposes of attribute identification. In these cases unqualified attribute references will be qualified by the current entity. For example, if the declaration is

```
DCL Y ENTITY, Z ENTITY CURRENT
(JOB) . . .
```

then Statement 34 could be written

```
FINISH = START + DURATION;
```

with the implication that this operation is to be done in the current job entity.

This is but one example of APL effectiveness. An extended version of this problem which organized the activities and printed out a table was written some time ago in NOMAD.¹³ The writing of the program took three weeks and required 500 statements. It is estimated that a 3 to 1 savings in programming time and a 5 to 1 savings in number of statements would have been realized, had the program been written in APL.

SUMMARY

The writing of programs in which the relation and dynamic manipulation of data elements is important can be made much easier with APL, a language designed at the General Motors Research Laboratories to fulfill this need. It closes the gap between algebraic languages and data handling needs with statements and defining capability permitting the description of data relationships. It further provides a means for efficiently expanding the programming environment with automatic file handling facilities.

APL statements are embedded in PL/I and are sifted out by a preprocessor. Thus, the user has a full PL/I capability in addition to the APL data handling statements. Output from the preprocessor is PL/I statements and subroutine calls.

Two new data types are introduced in APL, the SET and the ENTITY. These have a semantic relationship with the existing data types. This relationship is further clarified by the use of SET and ENTITY variables in the APL statements and control phrases.

The APL preprocessor is not a simple macro expander. The new data types, the entity declarations, the expanded use of the "." data qualifier and the expanded Boolean tests call for a processor. The processor tabulates information from the declarations and data types and later references the tables to produce an efficient symbolic PL/I output.

One goal of a high level language is to make computerized problem solving easier. By using APL commands the programmer is able to work with the data in terms of function and not in terms of the petty details and the pointer manipulations which actually perform the task.

ACKNOWLEDGMENTS

The design of APL was made possible only through the efforts of many members of the staff at the General Motors Research Laboratories. One of the first evaluations of APL was made by John T. Murray whose subsequent assistance in defining the language merits specific mention.

REFERENCES

1. J. F. Nolan and A. E. Armenti, "An Experimental On-Line Data Storage and Retrieval System," Technical Report 377, Massachusetts Institute of Technology Lincoln Laboratory, February, 1965.
2. E. Bennett, et al, "AESOP, A Prototype For On-Line User Control of Organizational Data Storage, Retrieval and Processing," *AFIPS, Volume 27, Proc. FJCC*, Spartan Books, Washington, D.C., 1965.
3. I. E. Sutherland, "Sketchpad, A Man-Machine Communication System," *AFIPS, Volume 23, Proc. SJCC*, Spartan Books, Washington, D.C., 1963.
4. M. J. Goldberg, "Network Analysis by Computer," *Instruments and Control Systems*, September, 1965.
5. E. L. Jacks, "A Laboratory for the Study of Graphical Man-Machine Communication," *AFIPS, Volume 26, Proc. FJCC*, Spartan Books, Washington, D.C., 1964.
6. "IBM Operating System/360 PL/I: Language Specifications," IBM Form C28-6571-2, IBM Corporation, White Plains, N.Y., 1966.
7. A. Newell, *Information Processing Language-V Manual*, Prentice-Hall, Englewood Cliffs, N.J., 1961.
8. *COMIT Programmers Reference Manual*, MIT Press, Cambridge, 1961.
9. J. McCarthy, et al, *LISP 1.5 Programmers Manual*, MIT Press, Cambridge, 1962.
10. J. Weizenbaum, "Symmetric List Processor," *Comm. ACM*, vol. 6, no. 9, (1963).
11. L. G. Roberts, "Graphical Communication and Control Languages," *Second Congress on the Information System Sciences*, Spartan Books, Washington, D.C., 1964.
12. J. B. Dennis, "Segmentation and Design of Multiprogrammed Computer System," *Journal ACM*, vol. 15, no. 4, (1965), p. 589.
13. *NOMAD Reference Manual*, General Motors Research Laboratories, Warren, Michigan, 1961.

AUTOMATIC OFF-LINE MULTIVARIATE DATA ANALYSIS

George S. Sebestyen

*Office of the Secretary of Defense
Washington, D.C.*

INTRODUCTION

Many research problems in the social and physical sciences require the collection of large amounts of data of the simultaneously measured attributes of a phenomenon or process under investigation. Pattern recognition problems, in particular, yield data of multiple variables for each manifestation of the different sources of data. The automatic off-line multivariate analysis techniques described in this paper deal with the quantitative description of data of this type.

There are two principal objectives in undertaking an off-line analysis of multivariate data. One of these is the objective of the researcher who studies the data in order to explain the characteristics of the data generation process. The study of the ocean as an acoustic propagation medium (through the analysis of waveforms received in response to excitations of known characteristics) exemplifies this research objective.

Another objective is that of the system designer who wishes to design a detection or pattern recognition system which must partition multivariate data into sets according to their sources of origin.

In either of the above applications a description of the nature of the data is necessary.

It is now common practice to regard each manifestation of a data source (or pattern class) as a

vector in a space of many dimensions, to consider the pattern class described by the joint probability density of its multiple variables, and the discrimination between pattern classes to require partitioning the vector space into regions associated with the different classes. For all these purposes, therefore, the principal objective of off-line data analysis is to describe the properties of the joint probability density of the multiple variables.

Taking the pattern recognition system design problem as the motivating force of this paper, a set of useful analyses, the reasons for their usefulness, and the manner in which the results would aid the system designer will be described.

DESIRABLE OFF-LINE ANALYSES

In the solution of practical pattern recognition problems the application of statistical analysis techniques to casually selected measurements is a poor substitute for expertise in the application and for care in the choice of simultaneous measurements. Even expertise, however, can only lead to a choice of measurements and hypotheses regarding their distributions. The analysis of collected data can be used to check the validity of hypotheses and to give rise to new ones.

The descriptors of the joint probability densities of measurements given below are useful in describing the data generation processes in terms that facilitate

the choice of pattern recognition techniques. They are first listed and then discussed.

1. The dynamic ranges of the variables.
2. The number and location of the modes.
3. The "irreducible" error, describing the lowest classification error that can be achieved with any method.
4. The degree of disjointedness of the regions occupied by members of different classes.
5. The size of the "clusters" associated with different modes.
6. The "distances" between modes.
7. The correlation between variables.
8. Assorted conventional statistics (mean, variance, etc.).
9. Various information theoretic measures (entropy, information gain, divergence).

The importance of knowledge of the dynamic ranges of variables is in providing guidelines for the resolution requirements of the classification system and in pointing to differences in resolution requirements between different variables. The performance of a classification technique employing the comparison of Euclidean distance measurements in several dimensions, for instance, would be completely dominated by the variable having the largest dynamic range, if large differences in dynamic range exist between different variables. This effect can be compensated for by normalizing the dynamic range of variables either through applying different scale factors to different variables or through the employment of a more general quadratic form which incorporates such scale factors.

Knowledge of the number of modes is needed in the selection of pattern recognition methods to avoid the use of techniques that assume distributions different from those that describe the data. Knowledge of the multimodal nature of the distributions, for instance, would tend to rule out the application of techniques that classify by cross-correlating the input with a stored reference or by evaluating one Gaussian probability density per class.

The "irreducible" error is a term coined to describe the probability of error that would be obtained if maximum likelihood ratio decisions were made and the joint densities and a priori probabilities were known. It represents the lowest error rate that could

be achieved, assuming exact knowledge of densities, and is thus a measure of the discriminability of the classes based on the chosen measurement set. If the irreducible error is not sufficiently low, a new set of measurements might be indicated. By computing this error on measurement subsets, a smaller set of measurements still yielding adequate performance can be selected.

It is of practical interest to know the amount of overlap between different classes in the measurement space. If the overlap is small, only the regions occupied by the different classes must be known. If the overlap is great, the values of the probability densities must also be correctly estimated to render the optimum decision.

Knowledge of the size of the clusters and distances between clusters is important in determining if simple techniques might be used. If the ratio of distances between clusters to the diameters of clusters is large, on the average, partitioning the vector space with a few linear discriminants may result in as low an error rate as that which would result from the use of techniques that attempt to approximate the probability densities of classes faithfully.

If several variables are highly correlated, a reduction can often be achieved in the number of variables needed. This should not be interpreted to imply, however, that it is necessarily desirable to deal with uncorrelated variables. The usual statistics of interest, means, correlation matrices and covariance matrices are of interest mostly only if the distributions are known to be gaussian, or at least unimodal. Their significance is reduced when the distributions are multimodal.

Information theoretic measures of entropy, information gain, and divergence are useful in measuring the spread of the distributions in the vector space, the value of different variables or sets of variables in describing the two classes, and in discriminating between classes.

The statistical measurements described above often facilitate the narrative description of the data and the generating processes in terms useful not only in the characterization of the data sources but also in the design of systems to discriminate between sources.

The descriptors of the joint probability densities listed above in (1), (7), and (8) are well known and their computation will not be dealt with here. Work on the determination of the number and location of modes from estimated joint probability densi-

ties is in progress and will be reported when the work is completed. The objective of successive sections of this paper is to describe techniques for obtaining the descriptors listed in (3), (4), and (9). With these descriptors the difficulty of the discrimination problem and the usefulness of different measurement subsets can be evaluated. Since the computation of all of these descriptors requires knowledge of the probability densities, first a method of approximating the densities of all data sources from available data of known classification but arbitrary order will be described.

PROBABILITY DENSITY ESTIMATION IN OFF-LINE DATA ANALYSIS

In on-line analysis techniques data must be processed in the sequence in which it is generated. By contrast, off-line techniques afford the opportunity to examine the entire data base simultaneously, permitting the repeated processing of data without regard for the time sequence in which it was collected.

An off-line probability density estimation process is described below for approximating the densities of several random processes (classes) by a single multidimensional histogram through the iterative examination of collected data. The technique is a simplified variant of an on-line "learning" technique described in Ref. 1. It differs from the latter in that here a single cell structure is generated to represent the probability densities of all classes, in the degree of supervision that can be provided by the analyst during density estimation, and in the fact that such supervision is made possible by the regular output of certain diagnostics with which the program informs the human analyst of the status of the density estimation process. The form of the output makes the computation of the previously listed descriptors of the classes particularly easy.

The program described below constructs a cell structure in the multidimensional space in which the cell locations, shapes, and sizes adapt to the analyzed data. The program can analyze data introduced consecutively or sorted according to a preestablished code. The number and identity of the variables analyzed is at the option of the analyst. Certain quantities useful from a diagnostic standpoint are printed out in various phases of the program to give the analyst the means for properly monitoring the probability density estimation process. The computer also carries out a limited dialogue with the analyst to provide the analyst with a printout of his available

options of action and to provide a permanent record of the course of action he took and the results that followed.

The program first types headings necessary for the proper identification of the problem. The analyst must fill in the blanks before the computer proceeds to the next step. A typical example of this initializing part of the program is shown in the printout of Fig. 1. In this example data meeting certain sorting requirements was to be used. A *one* in the "identifying mask" is used to designate the bit positions of the "identifying code" used in sorting vectors. The number and choice of variables is identified under the heading "initial starting cell size". The number of vectors to be processed and the minimum size of cells and "guard zones" to be used is chosen by the analyst according to methods described in Ref. 1. The significance of these analyst choices and program parameters will be explained below. The frequency of required diagnostic outputs is selected, and the identification of the data source (punched paper tape, magnetic tape, electrical signals) as well as its bit configuration (BCD or binary) is designated.

During the probability density estimation phase of multivariate data analysis it is assumed that each input vector contains a class designation and an identifying code.

The description of the probability density estimation program is illustrated by the flow chart of Fig. 2.

```

JOB NUMBER 1
DATE: APRIL 11, 1965
PROBLEM DESCRIPTION 17# - 2 DIMENSIONAL VECTORS

INITIALIZATION
SHOULD CONSECUTIVE DATA SAMPLES BE USED NO
IDENTIFYING MASK IS 11#00000000#
IDENTIFYING CODE IS 11#00000000#
IS INPUT BCD YES
NUMBER OF SAMPLES TO BE USED = 17#
THRESHOLD = 3.00#
GUARD ZONE = 15.00#
INITIAL STARTING CELL SIZE
DIMENSION
1 2.00#
2 2.00#
NUMBER OF SAMPLES BEFORE CALCULATING DIAGNOSTICS = 25

SET SW1 TO ON POSITION TO TYPE DIAGNOSTICS.
LOAD READER OR MAG. TAPE UNIT WITH INPUT DATA.
PRESS START TO BEGIN

```

Figure 1. Initialization in probability density estimation.

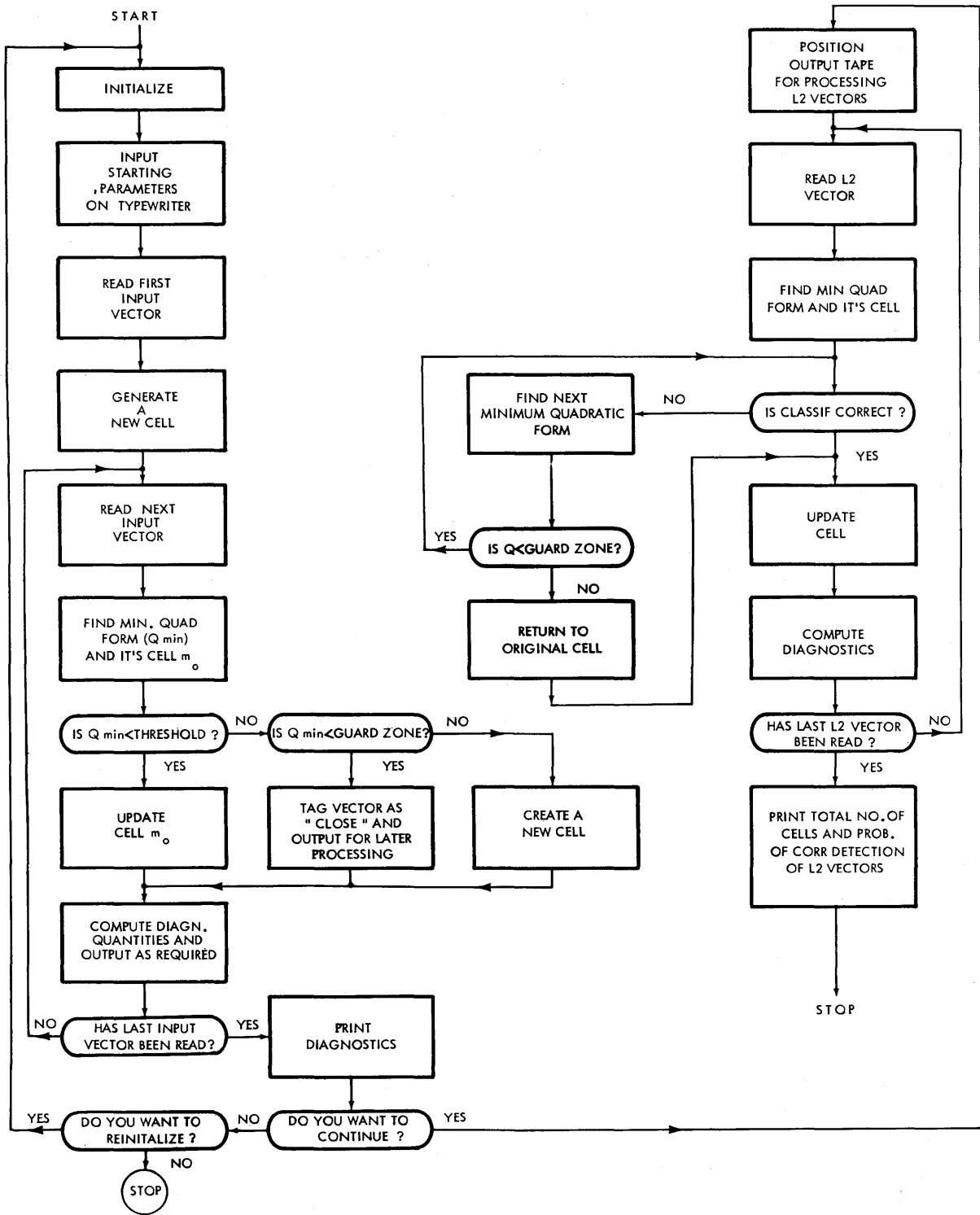


Figure 2. Simplified flow chart of probability density estimation.

The first input vector that passes the sorting step generates the first cell. The vector becomes the cell center, and the cell dimensions are determined by the inequality given in Eq. (1), in which the $\sigma_n(0)$'s and T (threshold) are obtained from the initializing information given by the analyst (see Fig. 1). Here v_n is the n th coordinate of an arbitrary point in an N -dimensional space, and v_1 is the n th coordinate of the first vector.* Contours equidistant from cell centers are ellipsoidal.

$$* \sigma_{1n}(0) = \sigma_n(0)$$

$$Q(v, v_1) = \sum_{n=1}^N \left(\frac{v_n - v_{1n}}{\sigma_{1n}(0)} \right)^2 < T \quad (1)$$

In addition to storing the cell center and cell radii, an estimate of the density of each class in the cell can be obtained by keeping track of the number of members (hits) from each class in each cell, and the volume of each cell.

After an input vector is introduced to the program (after it passed the sorting step), it is first treated as if its classification were unknown. In reality its classification is known. The input vector is compared with all stored cell centers by means of their respective quadratic forms; and the smallest quadratic form, Q_{min} , and the corresponding cell center, m_0 , are found. Since a histogram in the form of hit counts from all classes is stored for each cell, the new input could be classified (if it were of unknown classification) by the class identity of the highest hit count in the cell containing the new input vector. A record of the classification is kept for later use as one of the diagnostic outputs. After the smallest quadratic form, Q_{min} , is determined, it is compared with the threshold, T (part of the input information), to determine how close the new input is to the closest already established cell center.

If Q_{min} is less than T (that is, if the input vector is contained in an existing cell), then the cell is updated. Updating consists of recomputing the vector mean of inputs falling in the cell, computing their sample variances at the present time $\sigma_{mn}(t)$, and choosing new cell radii to equal, whichever are larger, the initial cell radii—determined by $\sigma_n(0)$ —or the cell radii determined by $\sigma_{mn}(t)$. After the cell has been undated, certain diagnostic quantities are computed. These will be described later.

If the input vector falls outside any of the existing cells (Q_{min} is greater than T), a check is made to determine if it falls in an annulus (or shell) surrounding the cell to which it is closest. This is done

by comparing Q_{min} with a guard zone threshold (see Fig. 1). If the vector falls in the guard zone, then the input vector is tagged as "close" to an existing cell but not within the cell. Tagged vectors are included in a set L_2 which is stored but not processed until a second pass through the program. The set L_2 contains all vectors set aside for later processing. In the particular program described here, members of L_2 are punched on paper tape.

If the input vector falls outside the guard zone, a new cell is created by storing the vector as the mean of the new cell, setting the cell hit count for the appropriate class to one, and setting the initial cell size to the initial cell radii, determined by $\sigma_n(0)$.

The analyst, in the meantime, is given the opportunity to observe the progress of the histogram construction procedure by a set of optional diagnostic outputs shown in the printout of Fig. 3. This printout, updated after every group of input vectors (the group size is specified during initialization), contains information described below. Assume, for the sake of illustration, that a diagnostic output is required after every 25 input vectors. The number of new cells (per 25 input vectors) is provided in the first column of the printout. In the second column the number of input vectors (per 25 inputs) that fell in guard zones is indicated. The probability of correct classification (averaged over the last 25 inputs) is indicated next. The cumulative number of processed input vectors is indicated in the last column.

In addition, every time a cell is updated, the serial number of the cell, the number of hits it now contains, and the location of the means and standard deviations (cell radii), truncated in the printout, are indicated for all of the vector components. A summary of the number of cells, the number of vectors in guard zones, and the total number of vectors processed is prepared at the end of the first pass through the data.

At this point the analyst is given an opportunity to check the diagnostic outputs to determine how to proceed next. The probability density estimation process is satisfactory if

1. The number of new cells created per diagnostic interval decreases with time. This indicates that the existing cell structure increasingly better represents the analyzed data.
2. The total number of cells created is a small fraction of the number of input

NEW CELLS	L2 VECTORS	PROB. CORR. DETECTION	TOTAL SAMPLES	UPDATED CELLS—LOCATION AND				STD. DEVIATIONS					
				CELL HITS	VC	S	VC	S	VC	S	VC	S	
15	10	.5598	25	10	2	27	1	11	1				
				6	2	13	1	33	1				
				11	2	17	0	38	1				
				11	3	18	1	38	1				
5	18	.9600	50	12	2	18	1	9	0				
				13	2	5	0	40	0				
0	19	.8799	75	13	3	5	0	39	1				
				1	2	42	1	46	0				
				9	2	32	1	41	0				
				16	2	11	1	48	0				
0	15	.8799	100	9	4	32	1	42	0				
				9	5	31	1	42	0				
				13	6	6	1	38	1				
0	15	.9199	125	12	5	19	1	9	1				
				11	5	17	1	38	1				
				3	2	44	0	9	1				
2	15	.9600	150	12	7	19	1	9	1				
				16	4	10	0	48	1				
				10	7	26	1	10	1				
0	12	.7998	170										
20	104	0.0000	170										

DO YOU WANT TO CONTINUE YES

POSITION OUTPUT TAPE TO READ L2 VECTORS. PRESS START TO CONTINUE

Figure 3. Probability density estimation program printout (Phase 1).

vectors. This indicates that the initial cell dimensions have probably been correct, for the cell generation process did not degenerate to the construction of a look-up table (by too small a choice of initial cell radii), and it did not create a few cells only (by too large a choice of initial cell radii).

3. The probability of correct detection (or classification) on inputs treated as unknowns (prior to their use in the density estimation process) should increase with time (or should have a sufficiently high absolute value). This is a measure of the classification performance that would be obtained if the density estimation process were terminated at any given time.
4. There should not be, in general, an unusual growth of any one cell. This can be verified by observing that the same cell number is not updated too frequently and does not contain a number

of hits out of proportion with other cells. In addition the cell radii (called standard deviations in Fig. 3) serve as direct measures of the cell size.*

Note that the above-mentioned desirable properties of the first phase of the probability density estimation process are reasonably well satisfied by the example given in Fig. 3. Also note that no significant cell growth has taken place as yet.

If the analyst is satisfied with the above-mentioned indicators of satisfactory progress of the estimation process, he can proceed to the next phase to be described below. If he is dissatisfied with the results, the analyst is asked if he wishes to reinitialize and start anew, perhaps with a different choice of initial cell dimensions.

If the analyst decides to go on, then vectors that fell into set L_2 are processed. The purpose of processing is to adapt the shape and size of already existing cells to the local distribution of data without

* Standard deviations in this printout were truncated at integer values.

increasing the storage requirements. The first L_2 vector is selected and tentatively classified by computing the smallest quadratic form relating the input vector to the stored cell centers and by determining the class designation of the highest hit count in the cell to which the vector is closest. If the tentative classification is correct (that is, the vector has the same classification as that associated with the nearest cell center), then the cell is updated, and the serial number, hit count, location, and size of the updated cell are printed out on an optional basis.

If the tentative classification of this vector is incorrect (if the vector is near a cell but the classification of that cell is not the same as that of the vector), then the search for another cell (second closest to the input vector) is continued to determine if it has the correct classification. If so, that cell is updated; if not, the search is continued for other cells still close enough to the input vector (determined by the criterion that the input lies in their guard zone). If there are none, the cell whose center is closest to the input vector is updated anyway. The purpose of this procedure is to stretch existing cells in such directions that the probability of correct identification of inputs should be maximized.

On an optional basis the updated cell numbers, their hit counts, locations, and dimensions are printed out. A portion of a printout of Phase 2 of the probability density estimation program is shown in Fig. 4. At the end of the printout a brief summary, stating the total number of cells and the probability of correct recognition of vectors of the set L_2 (treated as unknowns) is given. It is seen from Fig. 4 that satisfactory performance would be obtained from the cell structure created by the data. The growth of some cells is also evident from this printout.

The probability density estimation program outputs a set of stored vectors (centers of M cells in N dimensions), a set of N "standard deviations" for each of the M cells (the cell dimensions), and a set of hit counts for each cell (indicating the number of members of each class contained in the cell).*

In general it is not possible to determine how good an approximation of the actual probability densities the above procedure obtains, for the actual densities are unknown. In the case of the problem for which the printouts in Figs. 1, 3, and 4 were given, a com-

* The cell dimensions and hit counts define the probability density. Storing these quantities, however, takes less memory.

UPDATED CELLS-LOCATION AND				STD. DEVIATIONS			
CELL	HITS	VC	S	VC	S	VC	S
16	5	10	0	47	2		
11	7	17	1	37	1		
4	3	24	1	20	1		
14	7	33	1	8	1		
12	8	19	1	10	2		
16	6	10	1	47	2		
14	8	34	1	7	1		
12	9	10	1	10	2		
5	4	22	1	43	1		
6	4	11	2	34	1		
19	2	43	0	38	2		
5	5	22	1	43	2		
10	8	27	2	11	1		
3	3	43	1	8	1		
10	7	10	1	47	2		
7	2	40	0	56	2		
9	6	32	2	42	0		
6	5	11	2	35	2		
20	2	17	1	53	1		
12	10	20	1	9	2		
5	6	21	2	43	1		
3	4	42	2	9	1		
10	9	26	1	10	1		
6	6	11	2	35	2		

TOTAL NUMBER OF CELLS = 20

PROB. OF CORR. DETECTION OF L_2 VECTORS = .9229

LEARNING COMPLETE

Figure 4. Probability density estimation program printout (Phase 2).

parison of the two-dimensional data, shown in Fig. 5, and the resulting cell structure, shown in Fig. 6 (with cell boundaries shown at the one-standard deviation contours), reveals that the cell structure approximates correctly the shapes and densities of the regions in which the data is contained.

AUTOMATIC ANALYSIS OF MULTIDIMENSIONAL HISTOGRAMS

When the number of dimensions precludes the analysis and description of probability densities by inspection, they may be analyzed by means of the descriptors listed earlier. The two programs described below compute the descriptors listed in (3), (4), and (9), respectively. The purpose of the first program is to analyze the N -dimensional histogram obtained by the method described in the preceding to determine the lowest achievable error probability, the percentage of the vector space occupied by each class, the degree of overlap between classes, and other quantities to be described below.

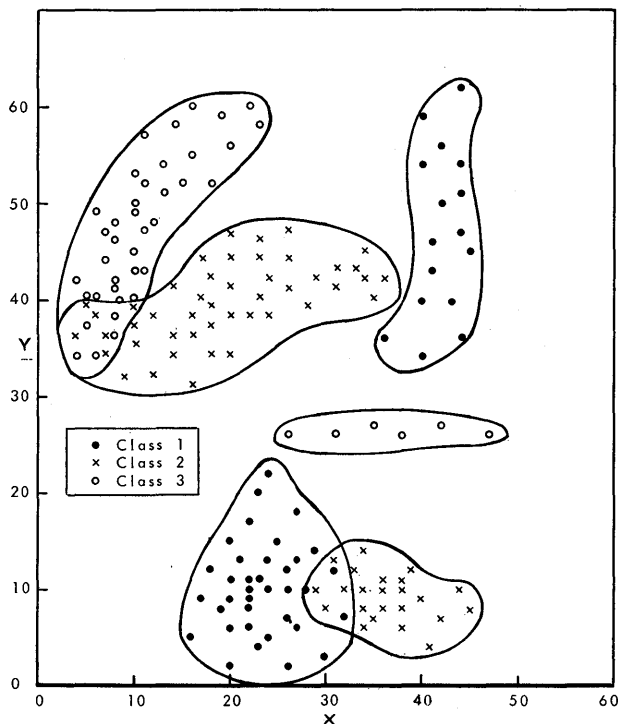


Figure 5. 170 two-dimensional vectors.

The following definitions will aid the description of the calculations performed.

- m = general index or serial number of a cell.
- M = total number of cells.
- X_{ci} = i th cell among those only containing members of class c .
- Y_{ci} = i th cell among those whose highest hit count is from class c .
- Z_{ci} = i th cell among those containing non-zero hit counts from class c .
- S_{cm} = hit count in cell m from class c .
- $PDCO$ = probability of detecting class c with probability of error = 0.
- PDC = probability of detecting class c .
- PER = probability of error.
- σ_{mn} = standard deviation of cell m in dimension n .
- M_c = total number of vectors of class c .

The program describes the distribution of the classes by calculating and printing the following quantities (an illustration of this printout is shown in Fig. 7):

1. The total number of cells generated, M . This output is directly obtainable from the probability density estimation program.

2. The fraction of cells containing only members of class c (repeated for all c). This output is obtained by summing the number of cells which contain hit counts only from class c .

3. $PDCO$, the probability of classifying members of class c with a probability of error equal to zero. Since the probability of error of a specific decision is zero if the likelihood ratio is infinite, the fraction of members of class c which are contained in cells only containing members of c (and thus having infinite likelihood ratios) is the quantity $PDCO$. Thus, $PDCO$ is given by Eq. (2). This quantity is computed for all classes.

$$PDCO = \frac{1}{\sum_m S_{cm}} \left[\sum_{\text{all } X_{ci}} S_{cX_{ci}} \right] \quad (2)$$

4. Probability of detecting class c , PDC , is measured by the sum of hit counts of cells in which the highest hit count is from class c . Thus PDC is given by Eq. (3).

$$PDC = \frac{1}{\sum_m S_{cm}} \left[\sum_{\text{all } Y_{ci}} S_{cY_{ci}} \right] \quad (3)$$

5. The irreducible error is the probability of error obtained when maximum likelihood ratio decisions are made using the best estimate of the probability

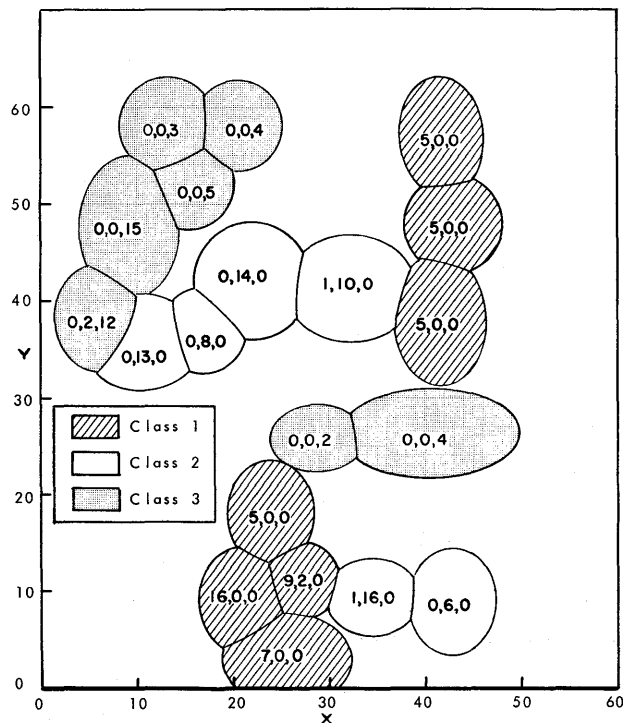


Figure 6. The cell structure.

JOB NUMBER 1
 DATE: APRIL 11, 1965
 PROBLEM DESCRIPTION 17 β - 2 DIMENSIONAL VECTORS

OUTPUT

TOTAL NUMBER OF CELLS GENERATED = 2 β
 TOTAL NUMBER OF HITS FOR ALL CLASSES = 17 β
 PERCENTAGE OF TOTAL VOLUME = . β 3 %
 NUMBER OF CELLS CONTAINING TIES = β
 FRACTION OF CELLS CONTAINING TIES = β

CLASS C	NUMBER OF HITS	FRACTION OF CELLS CONTAINING ONLY CLASS C	PROBABILITY OF DETECTING CLASS C W/ZERO ERROR	PROBABILITY OF DETECTING CLASS C	PROBABILITY OF ERROR	% OF VOLUME
1	54	.299	.795	.963	. β 37	. β 2
2	71	.199	.576	.943	. β 57	. β 2
3	45	.299	.732	1. β β β	β . β	β . β β

OUTPUT COMPLETE

Figure 7. Description of the distributions.

densities. The probability of error, *PE*, is given in Eq. (4).

$$PE = 1 - \sum_c PDC \quad (4)$$

6. The fraction of the volume of the space occupied by class *c* is the ratio of the sum of volumes of cells containing members of *c* to the product of the dynamic ranges of variables. This fractional volume, *VOL*, is given by Eq. (5). This is computed for all classes.

$$VOL = \frac{\sum_{Z_{ci}} (\text{volume of cell } Z_{ci})^* \times 1}{\prod_n (\text{max-min value of variable } n)} \quad (5)$$

7. The total fractional volume of the space occupied by all inputs is computed similarly, except the summation in Eq. (5) is carried out over all cells.

The information theoretic descriptors of entropy, information gain, and divergence can be computed readily from their respective integrals, given in Eqs.

* The volume of cell *Z_{ci}* is proportional to the product of the cell radii.

(6a, b, and c) with the aid of the cell structure obtained from the probability density estimation program.

$$\text{Entropy of class } c = H_c = - \int_{-\infty}^{\infty} p_c(v) \log p_c(v) dv \quad (6 a)$$

$$\text{Information gain } I_c = \int_{-\infty}^{\infty} p_c(v) \log \frac{p_c(v)}{p(v)} dv \quad (6 b)$$

$$\text{Divergence } J(i, j) = \int_{-\infty}^{\infty} [p_i(v) - p_j(v)] \log \frac{p_i(v)}{p_j(v)} dv \quad (6 c)$$

Each integral can be expressed as the sum of integrals over the *M* cells of the cell structure, and each integral over a cell can be expressed approximately in terms of the hit counts and dimensions of the cell. By this method the three quantities given above are expressed arithmetically in Eq. (7).

$$\text{Entropy } H_c = - \sum_{m=1}^M \frac{S_{cm}}{M_c} \log_e \left(\frac{S_{cm}/M_c}{N \prod_{n=1}^2 \sigma_{mn}} \right) \quad (7 a)$$

Information Gain $I_c =$

$$\sum_{m=1}^M \frac{S_{cm}}{M_c} \log_e \left[\frac{S_{cm}}{\sum_{n=1}^N 2\sigma_{mn}} \right] \left(\frac{S_{cm}}{\prod_{n=1}^N 2\sigma_{mn}} \right) \quad (7b)$$

Divergence $J(i, j) =$

$$\sum_{m=1}^M \left(\frac{S_{im}}{M_i} - \frac{S_{jm}}{M_j} \right) \log_e \frac{S_{im}/M_i}{S_{jm}/M_j} \quad (7c)$$

A printout from this program is shown in Fig. 8. Since the data sorting and computational steps of both of the above programs follow readily from the equations given, the flow charts will not be shown here.

SUMMARY

The set of analyses performed on multivariate data described in the preceding provide a rudimentary description of the data generation processes in terms that forecast the storage requirements (the number of cells) and the degree of discrimination that can be achieved between the processes.

The features of the probability density estimation technique which permit the intervention of the man in the machine analysis of the data and which provide the man with diagnostic outputs are important

CALCULATION OF ENTROPY, INFORMATION GAIN AND DIVERGENCE

JOB NUMBER 1
DATE: APRIL 11, 1965
PROBLEM DESCRIPTION 17β - 2 DIMENSIONAL VECTORS

INPUT

CLASS 1,2,3,

CLASSES 1-2,1-3,2-3,

PRESS START FOR OUTPUT

CLASS	ENTROPY	INFORMATION GAIN
1	3.6169	1.96β4
2	3.752β	1.9287
3	3.397β	1.7558

CLASSES	DIVERGENCE
1- 2	5.6β99
1- 3	2.521β
2- 3	4.4587

OUTPUT COMPLETE

Figure 8. Printout of entropy, information gain and divergence.

when a new application in which nothing is known about the data generation processes is tackled.

The multidimensional histogram representation of data simplifies the approximation of many complex analyses which would require integration over domains. The full potential of the multidimensional histogram representation of data has not been exhausted by the analyses described above.

REFERENCE

1. G. Sebestyen and J. Edie, "Pattern Recognition Research", AFCRL Report 64-821 (June 14, 1964).

DATA ANALYSIS AND STATISTICS: AN EXPOSITORY OVERVIEW *

J. W. Tukey and M. B. Wilk

*Princeton University and
Bell Telephone Laboratories, Inc.
Princeton and Murray Hill, New Jersey*

INTRODUCTION

Data analysis is not a new subject. It has accompanied productive experimentation and observation for hundreds of years. At times, as in the work of Kepler, it has produced dramatic results.

As in any other science, what is done in data analysis is very much a product of each day's technology. Every technological development of major relevance—organized tables of functions, knowledge of the mathematical consequences of the Gaussian law of error, desk calculators, stored-program electronic computers, graphical display facilities—has been accompanied by a tendency to rediscover the importance and to reformulate the nature of data analysis.

Today, as in the past, data analysis is usually difficult, cumbersome, and complex—and often very time-consuming, both in man-hours and in elapsed time. It is also of mushrooming importance in business, politics, science and technology.

The basic general intent of data analysis is simply stated: to seek through a body of data for interesting relationships and information and to exhibit the results in such a way as to make them recognizable

to the data analyzer and recordable for posterity. Its creative task is to be productively descriptive, with as much attention as possible to previous knowledge, and thus to contribute to the mysterious process called insight.

Four major influences act on data analysis today:

1. The formal theories of statistics.
2. Accelerating developments in computers and display devices.
3. The challenge, in many fields, of more and ever larger bodies of data.
4. The emphasis on quantification in an ever wider variety of disciplines.

The last few decades have seen the rise of formal theories of statistics, "legitimizing" variation by confining it by assumption to random sampling, often assumed to involve tightly specified distributions (in which a bare minimum of adjustable constants deny almost all flexibility) and restoring the appearance of security by emphasizing narrowly optimized techniques and claiming to make statements with "known" probabilities of error. While many of the influences of statistical theory on data analysis have been helpful, some have not.

Exposure, the effective laying open of the data to display the unanticipated, is to us a major portion of data analysis. Formal statistics has given almost no

* Prepared in part in connection with research at Princeton University sponsored by the Army Research Office (Durham).

guidance to exposure; indeed, it is not clear how the informality and flexibility appropriate to the exploratory character of exposure can be fitted into any of the structures of formal statistics so far proposed.

Many bodies of data require routine handling, not data analysis, and can be rightly approached with specific narrow questions. In dealing with them, some facets of formal statistics are more nearly appropriate, and have proved much more useful.

As a discipline, data analysis is a very difficult field. It must adapt itself to what people can and need to do with data. In the sense that biology is more complex than physics, and the behavioral sciences are more complex than either, it is likely that the general problems of data analysis are more complex than those of all three. It is too much to ask for close and effective guidance for data analysis from *any* highly formalized structure, either now or in the near future.

Data analysis can gain much from formal statistics, but only if the connection is kept adequately loose.

The impact on data analysis of the availability and capacity of computer hardware and software has already been substantial, but the full harnessing of this potential has hardly begun. Slowness in development of appropriate computing systems reflects the difficulty of the rethinking and restructuring of the science and art of data analysis which needs to be done.

The revolutionary computer and display developments now taking place will inevitably stimulate major extensions and departures in data analysis, whose beginnings are already visible. Today's first task is not to invent wholly new techniques, though these are needed. Rather we need most vitally to recognize and reorganize the essentials of old techniques, to make easy their assembly in new ways, and to modify their external appearances to fit the new opportunities.

Data has typically been easier to gather than to analyze, though there are outstanding exceptions. Despite the gains in computation and display—perhaps because of them—this is increasingly true. In so many fields, the accumulation of large volumes of data is becoming irresistibly practical and economical. As in the past, much, perhaps most, of even carefully collected data will not be adequately analyzed. In part this is because facts are usually more complex than the hopes which have led to their ac-

cumulation; in part because the accumulation of data dampens experimental excitement; in part because collecting data simply serves to keep the experimenter busy while he designs a more adequate experimental setup or develops a needed point of view; but also, in part, because the technology of data analysis is still unsystematized and many of those who could put its tools to good use are unable to do so effectively.

While the data-analysis facilities of the present and the foreseeable future entirely dwarf those of even the near past, it is apparent that the challenge to data analysis is growing instead of receding. Thirty years ago, many thought that good data-analytical techniques were really needed only when data was sparse. Today we also recognize that only the best data analysis will suffice when the data is very extensive. As bodies of data grow in size, the number of essentially different ways of approaching them increases, and the effort involved in their analysis threatens to grow even faster. The analysis of mass data challenges all of our ingenuity and resourcefulness. Meeting this challenge will also help us to handle small amounts of data more effectively and thoroughly.

Increasingly, most disciplines are evolving towards increased quantification and mathematization. Many of them (e.g., medicine) have had long histories of being descriptive and relatively qualitative largely because of the complexities of their phenomena and systems. In these, the demand on data-analysis techniques is often not only that they function in some sort of real time, but even that they perform as well as current expert judgment (which has often been developed over generations and perhaps is based on "data" still unrecognized). The resulting challenges to data analysis are major and increasing.

The wider variety of problems thus brought to data analysis increases the importance of recognizing general elements in diverse problems and untangling these elements from more specific ones. The need is to recognize and understand the similarities and differences of data analyses in nuclear physics, in the physiology of cell nuclei, in cloud-seeding, in the assay of antiviral agents, in chemical engineering, and in opinion polling, to select but a few.

DATA ANALYSIS IS LIKE DOING EXPERIMENTS

Far too many people, in the past and even in the present, have persisted in regarding statistics, and

even data analysis, as a branch of probability theory, nestled deep within modern mathematics. Happily this view is increasingly out of favor. Statistical data analysis is much more appropriately associated with the sciences and with the experimental process in general.

The general purposes of conducting experiments and analyzing data match, point by point. For experimentation, these purposes include (1) more adequate description of experience and quantification of some areas of knowledge; (2) discovery or invention of new phenomena and relations; (3) confirmation, or labeling for change, of previous assumptions, expectations, and hypotheses; (4) generation of ideas for further useful experiments; and (5) keeping the experimenter relatively occupied while he thinks.

Comparable objectives in data analysis are (1) to achieve more specific description of what is loosely known or suspected; (2) to find unanticipated aspects in the data, and to suggest unthought-of models for the data's summarization and exposure; (3) to employ the data to assess the (always incomplete) adequacy of a contemplated model; (4) to provide both incentives and guidance for further analysis of the data; and (5) to keep the investigator usefully stimulated while he absorbs the feeling of his data and considers what to do next.

Among the important characteristics shared by data analysis and the experimental process are these:

1. Some prior presumed structure, some guidance, some objectives, in short some ideas of a model, are virtually essential, yet these must not be taken too seriously. Models must be used but must never be believed. As T. C. Chamberlain¹ said, "Science is the holding of multiple working hypotheses."

2. Our approach needs to be multifaceted and open-minded. In data analysis as in experimentation, discovery is usually more exciting and sometimes much more important than confirmation.

3. It is valuable to construct techniques that are likely to reveal such complications as assumptions whose consequences are inappropriate in a specific instance, numerical inaccuracies, or difficulties of interpretation of what is found.

4. In both good data analysis and good experimentation, the findings often appear to be obvious—but generally only after the fact.

5. It is often more productive to begin by obtaining and trying to explain specific findings, rather than by attempting to catalog all possible findings and explanations.

6. While detailed deduction of anticipated consequences is likely to be useful when two or more models are to be compared, it is often more productive to study the results before carrying out these detailed deductions.

7. There is a great need to do obvious things quickly and routinely, but with care and thoroughness.

8. Insightfulness is generally more important than so-called objectivity. Requirements for specifiable probabilities of error must not prevent repeated analysis of data, just as requirements for impossibly perfect controls are not allowed to bring experimentation to a halt.

9. Interaction, feedback, trial and error are all essential; convenience is dramatically helpful.

10. There can be great gains from adding sophistication and ingenuity—subtle concepts, complicated experimental setups, robust models, delicate electronic devices, fast or accurate algorithms—to our kit of tools, just so long as simpler and more obvious approaches are not neglected.

11. Finally, most of the work actually done turns out to be inconsequential, uninteresting, or of no operational value. Yet it is an essential aspect of both processes to recognize and accept this feature, with its momentary embarrassments and disappointments. A broad perspective on objectives and unexpected difficulties is often required to muster the necessary persistence.

In summary, data analysis, like experimentation, must be considered as an open-ended, highly interactive, iterative process, whose actual steps are selected segments of a stubbornly branching, tree-like pattern of possible actions.

DATA ANALYSIS IN RELATION TO PEOPLE

The science and art of data analysis concerns the process of learning from quantitative records of experience. By its very nature it exists in relation to people. Thus, the techniques and the technology of data analysis must be harnessed to suit human requirements and talents. Some implications for effective data analysis are: (1) that it is essential to have convenience of interaction of people and interme-

diate results and (2) that at all stages of data analysis the nature and detail of output, both actual and potential, need to be matched to the capabilities of the people who use it and want it.

Data analysis must be iterative to be effective. Human judgment is needed at almost every stage. (We may be able to mechanize an occasional judgment.) Unless this judgment is based on good information about what has been found, and is free to call next for what is now indicated, there cannot be really effective progress.

Nothing—not the careful logic of mathematics, not statistical models and theories, not the awesome arithmetic power of modern computers—nothing can substitute here for the flexibility of the informed human mind. Accordingly, both approaches and techniques need to be structured so as to facilitate human involvement and intervention.

It is insufficient to have results produced; they must be displayed in a manner to satisfy diverse needs of a broad spectrum of individuals. The general goals include (1) using a body of data to answer specific questions, either formulated in advance or developed during the analysis; (2) exposing to people information and structure in the data that is of an unanticipated nature; and (3) compressed summarization for record and communication. All three require communication to people. For each of these purposes, the use of pictures is invaluable, both for guidance at intermediate stages and for final communication or summary.

In many instances, a picture is indeed worth a thousand words. To make this true in more diverse circumstances, much more creative effort is needed to pictorialize the output from data analysis. Naive pictures are often extremely helpful, but more sophisticated pictures can be both simple and even more informative.

For humans, the use of appropriate pictures often offers the possibility of great flexibility all along a scale from broad summary to fine detail, since pictures can be viewed in so many different ways. Moreover, one can change his approach from summary toward detail with great ease and great speed.

A few people will want to understand as much as they can about a given body of data. For them, both numerical detail and many pictures are useful. Others might like to know so much about many bodies of data, that they will have to be content with summarizations, which may often have to be extreme and accordingly somewhat misleading. Data

analysis must satisfy needs for both extremes, and, as well, for intermediate degrees of detail. (Formally, and historically, the emphasis has been too much on summarization.)

THE KEY TO EFFECTIVE DATA ANALYSIS

The iterative and interactive interplay of summarizing by fit and exposing by residuals is vital to effective data analysis. Summarizing and exposing are complementary and pervasive.

If certain aspects of the data have been effectively summarized by fitting a straight line, then we can improve exposure by building on this summarization. The plot of y against x is likely to be dominated by what has been summarized by the straight line. A plot of the residuals from the fit against x (or other variables) exposes to view only what has not been summarized, thus avoiding unnecessary distraction and permitting attention to finer details.

Even when used for confirmation alone, data analysis is a process of first summarizing according to the hypothesized model and then exposing what remains, in a cogent way, as a basis for judging the adequacy of this model or the precision of this summary, or both.

Techniques for summarizing are, fortunately, often useful for exposing and vice versa. For example, a half-normal plot² of contrasts in a 2^n experiment may serve as an effective summary of the data, with identification of interesting effects. It may also serve as an exposing technique in indicating, for instance, the unanticipated existence of two error terms.

This process of summarizing and exposing is intrinsically iterative. No step is clearly the last before it is taken.

Summarizing data is a process of constrained and partial description—a process that essentially and inevitably corresponds to some sort of fitting, though it need not necessarily involve formal criteria or well-defined computations.

To fit a straight line is to select one of a very restricted family of formal descriptions (a family involving only two constants) and to regard the line, or some coefficients that specify it, as a partial description of the data. To fit row and column means to a 2-way table, or to fit 9 main effects and 5 2-factor interactions to the data of a 2^{9-2} fractional factorial experiment, is to do something entirely similar,

differing mainly in the number of adjustable constants.

A process that is closely similar to these examples is drawing (freehand) a curve that is both increasing and "smooth" and that graduates some data reasonably well. There is, so far as we know, no finite set of adjustable constants that describe all "smooth" curves, but the family of acceptable curves is surely constrained. There is no precisely specified way to conduct the fit, but the result must surely be interpreted as a partial description.

Recognition of the iterative character of the relationship of exposing and summarizing makes it clear that there is usually much value in fitting, even if what is fitted is neither believed nor satisfactorily close. What is left over after the partial description from fitting can often be more effectively approached and structured because there has been some fit, even a poor one.

USING RESIDUALS EFFECTIVELY

When a "fit" has a sufficiently arithmetic character, there is a natural way to express what the fit has not described. Additive residuals defined by

$$\text{observation} = \text{fit} + \text{residual}$$

are widely used.

In other circumstances, it may be appropriate to express residuals in still other ways. After all, we are concerned with appropriate measures of deviation at each of several or many places, and there are many reasons why the appropriate way to measure deviation may vary from place to place, as well as from example to example. Multiplicative residuals or residual factors defined by

$$\text{observation} = \text{fit} \times \text{residual factor}$$

are sometimes useful, but are usually easily reduced to additive residuals by the taking of logarithms. If we are concerned with outlines of leaves, it may be reasonable to measure the deviation of actual outline from fitted outline at each of a number of places, probably at right angles to the fitted outline. And where observation of angles leaves a multiple of 2π undetermined, residuals are often usefully expressed by the sine of the difference between observed and fitted.

Adequate examination of residuals is one of the truly incisive tools of exposure. Perhaps because of the blinding effects of unrealistic optimism about as-

sumptions, perhaps because of the difficulties of computational practice during the era before modern computing evolved, and to an unfortunate degree because computer centers have felt that data output should be compressed, residuals have not received the attention and use they richly deserve.

There is no substitute for examining the collection of detailed individual residuals in diverse ways. It is almost always a sad inadequacy (though far better than nothing) to try to summarize the exposing information in a body of residuals by a mean square error. Even the computation of the individual residuals and their examination as an unstructured mass is not enough. Several graphs of residuals are usually in order. Related numerical analyses, which answer specific questions more stringently but more general questions hardly at all, can also be useful.^{3,4}

Kinds of plots of residuals that are very often valuable include (1) plots against fitted (or observed) values; (2) plots against variables which were employed in the summarizing fit; (3) plots against variables not used in the fit (e.g., time); (4) probability plots of ordered residuals, particularly plots of empirical quantiles against quantiles of reference distributions, such as the unit normal.

The first three kinds of plots are often effective in showing what changes in style of fit are needed. Probability plots are particularly helpful in indicating a few peculiar values and in illuminating the overall success of the fit. They provide quick information about location, about spread, about distributional peculiarities, and a palatable summary of individual residual values. In any residual plot it will be helpful to identify each individual residual according to whether or not it comes from an observation which was used in developing the fit. There should be an effort to identify and make evident other important qualitative characteristics of individual residuals.

The usefulness of residuals as a means of exposure depends on the summarizing model having identifiable deficiencies. Accordingly, residuals may fail to reveal deficiencies of a summarizing model when these are too varied and general, or when the data points are few in number or badly distributed.

An example of this is the behavior of residuals from an additive fit in a two-way/classification table. If the departure from additivity is due to the existence of just one highly deviant cell in the table, examining the residuals as a whole will tend to expose this cell. If there are two such deviant values, their

effects can combine to conceal any obvious peculiarities⁵ so long as the individual residuals are examined as an unstructured set of numbers.

THE STRATEGY OF DATA ANALYSIS

In addition to the two-pronged use of summarization and exposure, including careful attention to residuals, three of the main strategies of data analysis are:

1. Graphical presentation.
2. Provision of flexibility in viewpoint and in facilities.
3. Intensive search for parsimony and simplicity, including careful reformation of variables and bending the data to fit simple techniques.

Some people are apparently able to absorb broad information from tables of numbers.⁶ Most of us can only appreciate matters with full insight by looking at graphical representations. For large-scale data analysis, there is really no alternative to plotting techniques, properly exploited. A picture is not merely worth a thousand words, it is much more likely to be scrutinized than words are to be read. Wisely used, graphical representation can be extremely effective in making large amounts of certain kinds of numerical information rapidly available to people.

Flexibility in viewpoint and in facilities must be built into both the general technology and the individual techniques of data analysis. We must have flexibility in the choice of a model for summarization, in the selection of the data to be employed in computing the summary, in choosing the fitting procedures to be used and in selecting the terms in which the variables are to be expressed. Flexibility in assembly and reassembly of techniques is crucial.

Using human judgment in selection, or cleaning-up, of the data by partial or complete suppression of apparently aberrant values is natural, sensible, and essential. Data is often dirty. Unless the dirt is either removed or decolorized, it can hide much that we would like to learn. Sometimes, it is true, the dirt is blue clay, and contains diamonds in the form of new phenomena and new insights. Whether it is worth much or little to prospect for diamonds among the consequences of a particular set of data, we can do this better after labeling as dirt whatever *appears* from analysis to be such. Moreover, whether or not

it is diamond-bearing, clearing out the dirt can do much to help us learn from the cleaner data.

Suppression may be complete and wholly human, as when we decide to exclude a particular set of observations from all computations. Suppression may be partial and wholly automatic, as when we use the median of a set of observations, or the mean of the values in the two center quarters of the empirical distribution. In practice, the processes of selection and suppression are mixed up, rather complex, and for all that quite essential.

Just because values have been suppressed in fitting is no reason for their residuals from the fit to be forgotten. Not every suppression will have suppressed mere dirt. Some will have suppressed clean data, others will have suppressed diamonds. We will never be wholly sure which is which, but calculating, looking at, and thinking about residuals from suppressed data often stimulates the further questions needed to help clear up the situation.

The importance of parsimony in data analysis can hardly be overstated. By parsimony we mean *both* the use of few numerical constants and *also* the avoidance of undue complexity of form in summarizing and displaying. The need for parsimony is both aesthetic and practical.

In general, parsimony and simplicity will be achieved in the summary description either at the price of inadequacy of description or at the price of complexity in the model or in the analysis. Typically those who insist on doing only primitive analyses must often be satisfied with complex—not parsimonious—summaries which often miss important points.

An additional value from parsimony is illustrated in the following idealized example: A cubic in x will always fit the particular numbers that make up our body of data somewhat more closely than a quadratic. This may easily be more a seeming than a truth. If y only differs from a quadratic function of x by fluctuations, independent from one x to another, the fitted quadratic will fit the *average* values of y given x more closely, on the average, than will the fitted cubic.

One further aspect of great strategic importance in data analysis involves the transformation, better called reformation, of variables. Especially insightful choices of modes of expression underlie much of physical science. Changing from raw values to their square roots or logarithms (or other appropriate function) before the data is analyzed is often aston-

ishingly effective. Equally important is the evolution and use of techniques of analysis by which the data itself may be employed to indicate useful transformations.

As a matter of general strategy we may note here that it is almost always easier, and usually better, to "unbend" data to fit known analysis techniques than to bend the techniques to fit the data. If the square root, or logarithm, or reciprocal behaves in a simpler way than the raw form, it is obviously unwise to work with the data in the form where its behavior is more complex. With enough effort we can probably bend any of our techniques of data analysis to work explicitly and effectively on the data in its raw form, but this effort is rarely justified.

FITTING, THE WORKHORSE OF DATA ANALYSIS, HAS VARIED OBJECTIVES

The single most important process of data analysis is fitting. It is helpful in summarizing, exposing and communicating. Each fit (1) gives a summary description, (2) provides a basis for exposure based on the residuals, and (3) may have the parsimony needed for effective communication.

Fitting inevitably raises questions concerning classes of models to be used, selection of criteria of fit, choices of mode of expression for observations, as well as questions of numerical and logical algorithms. The answers to all these questions depend upon the diversity of the objectives of fitting, their character, and the differences amongst them.

These objectives include:

1. *Pure description*, in the sense of drawing, possibly hastily, a curve across the page and saying y appears to depend on x just about this way. If this is our only aim, we do want the curve to fit well, but we do not care at all whether its functional form is more than an accident. Finding, for instance, that a cubic polynomial fits our data well enough is not, at this level, to be thought of as giving any particular support for a cubic "law."

2. *Local prediction*, in the sense that, so long as the situation "remains the same," we should like to do well by substituting x 's into the fit and regarding the result as predicting the value of y . This amounts to hoping that our description of the past, however empirical, will continue to be a good description of the future.

3. *Global prediction of local change*, in the sense that we can use our fit to assess the result (averaged over fluctuations) of changing one or more x 's moderately, even when both the start and the finish of this change are far from the circumstances for which the fit was developed. If this is to be accomplished successfully, the general situation must be favorable, and theory, or insight, or broad experience must have been responsible for choosing the form of the fit and the nature of the y variables; the data before us can rarely be used to narrow things down enough to provide such good prediction, even of changes, elsewhere.

4. *Global prediction of values*, in the sense that we can use our fit to predict y given x far outside the range of the data on which it was based. Reliance upon outside information (including insight) is now even greater, and the chances of success are correspondingly diminished.

5. Using a fit depending on several mathematical variables (some of which may be functions of the same physical variable) to tell us *which variables have influences and which do not* (which can include telling us about the forms of the dependencies). This is sometimes possible, but nowhere nearly as often as is commonly hoped. Very frequently several alternative sets of variables will each give a satisfactory fit.

6. Using the fit to *estimate coefficients* having the general character of physical constants. Careful descriptions of both what is to be *varied* and what is to be held *constant* are essential before there is any hope of doing this effectively. "Heat capacity," for example, is not an adequate name. Heat capacity at constant volume differs substantially, both in meaning and value, from heat capacity at constant pressure. In most circumstances, indeed, constants to be assessed are not even as simply defined as heat capacity, rather they are only defined in terms of specific, rather complex functional forms.

These six objectives center on what has been fitted rather than on the other essential ingredient of fitting, what remains after the fit. Residuals have two quite distinct sorts of uses. On the one hand, they can be used as an immediate basis for further summarization, as in:

7. Providing *adjusted values for further study*, as when economic series are seasonally adjusted, or when the analysis of covariance is applied.

8. Providing a basis for immediate further fitting, as when the residuals from an eye-fitted straight line are fitted by either a further straight line or a quadratic. (So-called stepwise regression procedures operate in this general way, though they tend to omit the calculation of actual residuals.)

The more usual objectives for residuals emphasize exposure and include:

9. Examining and exposing with a view to learning about the *inadequacy of the fit*.

10. Examining and exposing with a view to identifying *peculiar values*, either for study in their own right or for suppression, partial or complete, from further analysis.

We need not assess the relative value and frequency of these specific objectives of fitting—the real need is to identify and distinguish varied objectives (of which these 10 are not all), to recognize their diversity, their tendency to occur one or a few at a time, and the consequent great variety of different demands made upon the fitting process itself and upon such associated procedures as plotting of residuals. No one fitting-and-residuals procedure can serve all our purposes.

REFORMATION OF VARIABLES

When is one expression better than another *for analysis*? Basically, when the data are more simply described, since this implies easier and more familiar manipulations during analysis and, even more to the point, easier and more thorough understanding of the results.

The usual goals of better expression include:

1. Additivity of effects
2. Constancy of variance
3. Normality (Gaussianity) of distribution
4. Linearity of relationship

Widespread and clear understanding of the relative desirability among the first three goals has been impeded by a happy fact—one that only appears accidental: All three tend to occur together. Where a choice has to be made among these three, additivity is to be preferred above all⁷ with constancy of variance second.

Linearity of relationship is important for both arithmetic manipulation and graphical presentation. If a response needs to be related to only one factor upon which it depends monotonely, a suitable

change of the expression of *either* the factor *or* the response will make the relationship linear. However, when as usual, we have to deal with two or more factors, we may be unable to reach linearity by any such simple device. In most such circumstances, linearity may be achieved if we can attain additivity since appropriate reexpression of *the factors* will then make the dependence of the response on them linear.

Some ways of seeking out desirable expressions are:

1. Explicit trial of various alternatives, whose evaluation is usually best done in terms of corresponding residuals.^{3, 8}
2. Use of numerical guides to the next choice.^{3, 4}
3. Use of computer iteration to seek that monotone change of expression which produces the greatest amount of additivity.⁹

All of these approaches work. Which one is desirable in a particular case depends upon objectives, on the amount and shape of the data, and on the availability of computing and display equipment.

The gains from appropriate reexpressing, transforming—more simply *reforming*—individual variables are likely to be substantial. More major gains (e.g., gains in efficiency by factors of 2, 3, or often much more) come from such efforts than from any other data-analytic step.

SCALING IN DATA ANALYSIS

Measures of similarity (or dissimilarity), even when expressed on a rubber scale, can be used to generate quantitative variables (see Refs. 10–14). Even starting with several responses, each expressed on a well-established numerical scale and thus able to serve as coordinates in a Cartesian space, these (and related) methods can sometimes generate non-linear transformations of the original responses from “distances” among the points in the initial space.

NEW LINEAR COMBINATIONS FOR OLD

Interest in replacing one set of variables with another made up of linear combinations of the first variables arises:

1. in preparation for dropping some of the new variables;

2. in order to make calculations involving these variables either simpler or more understandable;
3. in a search for a more meaningful or insightful coordinate system.

Canonical Analysis

The computations classically used for calculating canonical correlation coefficients and canonical variates can be used in much more general situations to provide an ordered family of linear combinations of the original working variables, guided by the ability of initial subsets of this family to describe, through linear regression, the behavior of one or more guide variables. Principal components are logically, but probably not computationally, just a particular case.

Orthogonalization

Recognition and elimination of close approximations to linear dependences is almost always important in three ways: computationally, descriptively, and conceptually. Direct quantitative description of amount of dependence upon factors that are substantially correlated with one another still appears almost hopeless, at least so far as communication to the human mind is concerned. Computational difficulties from unrecognized near linear dependencies can be very great. Changes of coordinates to avoid such problems are often very useful.

Complete elimination of correlation—precise orthogonality—is of little consequence to us, so long as our arithmetic processes do not assume it. Still, in practice, we usually seek “orthogonality,” mainly because it is specific, clearly defined, and related to simple algorithms. In particular, variables made orthogonal for one set of data are often thoroughly useful in analyzing another, where they are only approximately orthogonal. This happens most frequently, perhaps, when the second set of data is a subset of the first, or, conversely, when the first set is a random sample of the second (as may often be computationally convenient in dealing with large bodies of data).

Orthogonal polynomials in other than the usual order may be useful. Orthogonalization of more general variables is also often both convenient and useful. Notice that canonical variates, pure or modified, (and thus, in particular, principal components) are automatically orthogonal.

Rotation

Rotation of an initial coordinate system to a more useful position can be helpful, but requires quite explicit information about the advantages and disadvantages of specific choices.

CONCENTRATING ON A SUBSET OF THE VARIABLES

Even in the simplest case of naturally or prescriptively ordered variables, working from limited and fallible data, as we always do, offers NO hope of always, or even usually, dividing the variables into ONLY two classes: those it appears we must keep, and those surely of no importance.

Since the “middle class” variables should usually *be carried on to further analysis*, it will almost always NOT be vitally important to be highly precise in just how we select a sequence of linear combinations, the first k of which we are to carry on to further analysis.

In the more general problem of choosing any subset of variables for retention, we face new problems. The 11th, 23rd and 47th variables may, for example, be so highly correlated with each other that any one can deputize for any other without appreciable loss. When this is so, no one can be essential, since it could be replaced; but neither can we be sure that any one of them is *not* important. Also, it is easy to produce a situation where either of two variables alone has negligible descriptive power, yet combined they are very effective.

Taken as a means of selecting reasonably satisfactory subsets and giving some indication of their comparative performance, procedures of stepwise, screening or “steered” regression can prove very helpful in many situations, particularly if the uncertainties and inadequacies of the objectives and the results are clearly recognized.

The basic idea of steered regression is iterative use of the following step: Ask how much can be gained (in the quality of one or more regressions) by adding each of the remaining variables to the currently selected subset, and then add to the subset that variable which gains the most. It is often, perhaps usually, useful to include another process in the iteration. Ask how little it costs to exclude from the subset each of the variables currently in it, and then, if the cost is low enough, remove the least costly variable from the subset. There also needs to be a stopping rule.

Conventionally, gains and costs are assessed in terms of changes in the residual sum of squares.¹⁵⁻¹⁷ More complex measures seem more reasonable in many, if not most, circumstances, however, and the evolution of steered regression is likely to involve more varied and insightful choices of steering functions.

GENERAL CONSIDERATIONS IN GRAPHICAL PRESENTATION

Graphical presentation appears to be at the very heart of insightful data analysis. For most people, graphs convey more of a message than tables and do so more persuasively and attractively. Graphical presentation continues to hold its preeminent place despite both feeble understanding of the reasons for its power and appeal and severe limitations on the variety and character of its techniques, the latter stemming both from past technological limitations and from continuing inadequacies of imagination. Why?

Some reasons are easily found: Graphical displays can be very flexible. The human eye and brain are speedy and proficient in recognizing certain types of geometric configurations. "Smoothness" seems to be very much a geometric concept. The eye seems much more able to comprehend nonunderstood graphs than nonunderstood numbers. Quite large volumes of data can be displayed economically and comprehensibly. And the same graph can transfer effectively either a very compressed summary or an extensive amount of detail, as well as many intermediate packages of information.

Before we discuss some of these reasons in more detail, one key point must be made. While it is often most helpful to "plot the data," this is rarely enough. We need also to "plot the results of analysis" as a routine matter. (There is often more analysis than there was data.)

The innate flexibility of graphical displays is many-sided. It is not merely that we can choose to do many different things graphically. If one expects (or only contemplates the possibility) that y is approximately linear in x , a simple plot will confirm this when it is so and be even more instructive when it is not. If one has no clear anticipation, the same simple plot is likely to reveal whichever one of many alternative structures appears to be present, even though these structures are nowhere collected in a

list. (We may, indeed, even doubt whether it is humanly possible to list them all.)

The human eye and brain join easily and speedily in recognizing straightness and certain kinds of smoothness, in assessing amounts of local roughness or local variability (especially when properly aided by a reference "curve"), and in judging the presence or absence of systematic deviation (when the reference curve is neither too steep nor too wiggly). With less precision and more effort, eye and brain can judge symmetry and circularity moderately well, and have a fair chance of recognizing that certain features occur in a roughly periodic way. Further, of basic importance though more difficult to verbalize, the human eye and brain can learn to recognize quite complex and varied configurations with surprising effectiveness.

Almost all graphical techniques correspond to one or more natural reference situations. Graphs are most effective when these conceptually simple situations produce simple configurations, *above all when reference situations produce straight lines.*

"Smoothness" seems to be an essentially geometrical concept for which we do not yet seem to have a reasonable analytical approximation. "Smooth" extrapolation and interpolation, especially with irregularly spaced data, continues to be easier and more persuasive when conducted and exhibited graphically rather than numerically.

One great virtue of good graphical representation is that it can serve to display clearly and effectively a message carried by quantities whose calculation or observation is far from simple. Many kinds of spectra, analog and numerical, illustrate this principle.

A scatter diagram with 100 or 500 points need not be more difficult to scan than one with 10 or 50. The same is often true with 1000 to 5000 points (possibly with 10,000 or 50,000). Tables of numbers simply cannot be expanded comparably without tremendous increases in difficulty of examination and understanding.

Similar problems and advantages occur in the even simpler case of an unstructured collection of single-number data. As the volume of data increases, it becomes very difficult to appreciate from a table even the most elementary properties of the collection, such as location, range or gaps. Yet simple graphical representations, as empirical cumulative distribution plots¹⁸ or, perhaps, even as sensibly constructed histograms, provide rapid, easy and insightful indication of many properties, both sum-

mary and detailed, and do so as conveniently for large bodies of data as for smaller samples.

A common and extremely effective human response to a scatter plot of y versus x is often to draft in a smooth “freehand” or “eyeball” curve as an aid to judging the data. (Doing this is a natural step toward summarizing and exposing—toward the complementary processes of fitting and inspection of residuals.) Despite the apparent ease—and substantial agreement—with which humans can do this, there does not yet exist any automatic procedure that does it at all satisfactorily.

The issues and problems of graphical presentation in data analysis need and deserve attention from many different angles, ranging from profound psychological questions to narrow technological ones. These challenges will be deepened by the evolution of facilities for graphical real-time interactions.

ONE-VARIABLE GRAPHS

Graphical portrayal of frequency distributions by bar charts and histograms can be improved in various directions. For comparing with a fitted curve, in particular, the use of hanging (or suspended) rootograms, in which heights are proportional to the square root of frequency and blocks are attached to the fitted curve (not the base line), can be a considerable improvement.¹⁹

KINDS OF TWO-VARIABLE GRAPHS

Point plots, linked plots, and curve plots are only three of several distinct styles for two-variable graphs. Point clouds, scatter displays and progressive patterns are associated with useful distinction among purposes. Regularity of spacing, frequency of wild observations, absence of changes in variability, and correlations among fluctuations are also important considerations.

LINKING-UP POINTS AND RELATED ISSUES

Whether or not the points of a progressive pattern are linked together by line (or curve) segments can significantly influence the usefulness of such a graph. Linking up is not likely to help unless the points are reasonably close together (in x) and reasonably uniformly spaced (in x) and the corresponding “spectrum” is not too flat.

THE NEED FOR VARIOUS MENTAL APPROACHES

Given an appropriate plot, we still need an appropriate attitude or “set” for the brain to take toward its message. In this regard, the degree and character of correlations among the fluctuations of the various points are particularly important.

THREE-VARIABLE GRAPHS

Visual presentation of $z = f(x,y)$ is far from easy, yet badly needed. Of three classes of possibilities—contours, families of cross sections, and isometric views—the first seems, so far, most likely to be effective, though direct-interaction graphical consoles may offer other possibilities.

GENERAL CHARACTERISTICS OF DATA ANALYSIS

In productive data analysis:

1. *Those who seek are more likely to find.*

Tight frameworks of probable inference demand advance specifications of both a model and a list of questions to be asked, followed by data collection and then by analysis intended to answer only the prechosen questions. A man who lived strictly by this paradigm would have a hard time in learning anything new.

Some may be uncomfortable in not having tight global probability-like measures to calibrate their optimism and pessimism, yet in thinking about this difficulty it is vital to remember that science has not required independent confirmation without reason. To have clear evidence that something was not chance in one single circumstance is feeble proof that it happens in general. The price of losing a crisp evaluation of the results for a single circumstance is thus never great.

The price of not looking around, on the other hand, is the loss of opportunity to have the data suggest new things. What price would be greater?

In a strictly confirmatory experiment, there is a clear place for a relatively narrow and constricted analysis. But even there, there is likely to be a basic need and responsibility for accompanying such an analysis with a careful look around for new suggestions.

2. *Flexibility in viewpoint and in facilities is essential for good data analysis.*

Data analysis is very much a bootstrap exercise. Our facilities and our attitudes must encourage flexibility: use of alternate models, choice of subsets of the data, choice of subsets of auxiliary or associated variables, choice of forms of expression of these variables, and of the data, choice of alternate criteria, both in fitting and in evaluating.

3. *Both exploration and description are major objectives of data analysis; for both reasons data analysis is intrinsically iterative.*

Both the search for insight and for the unanticipated require that the available information be displayed. *Description as a preparation to display and insight is, in a certain sense, the main business of data analysis.* But, equally, adequate insight, however informal or intuitive, is a necessary precursor for incisive description of the anticipated. Accordingly, insightful exploration and description require an iterative, interactive, complementary process involving both summarization and exposure.

SIMPLICITY

Simplicity is in the mind, and it is valuable because it lets the mind work better. *Simplicity is often learned and comes in many forms.* To a man who understands only straight lines, a parabola may seem complex. As a conic section, however, it has a simplicity that has attracted men's minds since the days of the Greeks.

But, in data analysis, just as in experimentation, *attaining the simple is often a complex task.* Head-on collision between two high-velocity atoms or ions is simple in concept, yet a colliding-rings particle accelerator is a real complexity. Producing a simply-portrayed description of a body of data may require a similar complexity both in arithmetic approach and in display.

Progress in science is a curious mixture: The simple becomes complex as we learn about the ifs and buts; the complex becomes simple as new generations, using new concepts, learn to regard new things as simple.

Thus, in data analysis, *useful results, including useful techniques, need to be made simple.* This requires a broad spectrum of appropriate concepts.

THE LIMITATIONS OF DATA ANALYSIS

Data analysis cannot make knowledge grow out of nothing or out of mere numbers, nor can it salvage or sanctify poor work. *It can only bring to our attention a combination of the content of the data with the knowledge and insight about its background which we must supply. Accordingly, validity and objectivity in data analysis is a dangerous myth.*

Developing models of one sort to aid in appreciating or assessing the performance of models of another sort (perhaps describing methods of analysis) may indeed be useful to the discipline of data analysis. Such theories of inference must, however, be taken only as a guidance, and kept from becoming impediments. Assumptions and theory are indispensable, but, in use, the focus of data-analysis techniques must be on the data and the analysis, with the theory aiding insight by providing alternative backdrops.

It seems too easy for some to believe that detailed assumptions can make the data tell much more, either qualitatively or quantitatively, than would otherwise be the case. But if these assumptions are unwarranted their consequences may be misleading.

For example, combining an unexamined assumption of additivity in a two-way table with a classical test for main effects may indicate the absence of statistical significance, yet an elementary examination of residuals or interactions may reveal important information.

Both the guidance and the conduct of data analysis demand approximation. The combination of individually useful approximations often fails to be useful, either because errors accumulate or because ranges of adequacy fail to overlap. Thus, when *simple, individually useful models or data-analytic steps are linked together*, it is essential, as scientists have long realized, to think about the scientific problem as a whole and to make empirical tests of the worth of the combined chain.

GUIDANCE AND MODELS

Data analysis cannot be effectively conducted without guidance: implicit and vague or detailed and explicit. Contemplation of raw observations with an empty mind, even when it is possible, is often hardly more beneficial than not studying them at all.

In the sense in which we here use the word "model"—a means of guidance without implication of belief or reality—all the structures that guide data analysis, however weak and nonspecific, are models—even when they are not explicitly mathematical. Without them we are almost certainly lost (and surely completely primitive); were we to accept them unquestioningly we would be equally lost in a different morass; taking them as limited guidance, we may, however, succeed in finding some of what the data conceals. As Francis Bacon so well said, "Truth arises more easily from error than from confusion."

Definiteness in detailing objectives and assumptions in a formal model can simplify mathematical problems and increase the simplicity and impact of the results reached. But tightness of detail usually forces such a formal model unnecessarily far away from the realities of the data-gathering situation, obscuring possibly important phenomena. Looser structures can often do as well in simplicity and clarity of results while retaining robustness and breadth. *Both for guidance and the encouragement of exploration, it is most desirable that models be loose and noncommittal*, thus encouraging diverse alternative working hypotheses.

Even as simple a problem as comparing the location of two samples illustrates these points. When our model includes assumptions of approximate equality of variance, absence of seriously aberrant observations, and close normality (Gaussianity) of distribution, we are likely to calculate Student's t and halt. If we admit that any or all of these assumptions may be false, we will know that we need to do much better. (So far as the narrow objective of location comparison is concerned, it may suffice to use a more robust modification of Student's t .)

In most circumstances, we will gain by broadening our interests, and supplementing either t -value by at least inquiring what the sample has to say about inequality of variances, presence of aberrant observations, or nonnormality of distribution. Exhibiting the two samples on a single normal probability plot will surely open our eyes in these directions and will even, occasionally, direct our attention to less anticipated phenomena. The gain from doing this is likely to be great.

Trying to answer questions concerning the adequacy of a model by the use of data may be interesting and valuable. *In data analysis, however, models and techniques are to be thought of and developed*

as assisting tools with the focus on the data. The models need not fit perfectly or even adequately to prove usefully insightful. We must never believe so deeply in any model as to constrain our insight.

Thus, for example, although the use of half-normal plotting² in analysis of 2^n experiments was suggested by a model combining equally distributed normal errors, simple factorial effects, and the "null" hypothesis that these latter effects vanish, such plots merely use these assumptions to provide a "backdrop" for exposure and remain both descriptive and instructive in most circumstances when the assumptions fail, even badly. Indeed, the plot itself may indicate or reveal the inappropriateness of the assumptions.

BRIEF COMMENTS ABOUT SOME CLASSICAL STATISTICAL PROCEDURES

Particularly in textbooks, statistical procedures are usually described as operational wholes (e.g., multiple linear regression), too often in terms of a fragmentary list of formal objectives (e.g., to estimate regression coefficients and/or test hypotheses about narrow aspects of the model). The main emphasis in statistical theory and in textbook presentations of methods has been on confirmation and summarization (e.g., the basing of multiple regression methods on linear hypothesis theory).

The development of techniques and concepts *useful for exposure* has had very little guidance from formal statistical theory. In actual practice, statistical methods embodied in such categories as experimental design, analysis of variance, multivariate analysis, time series analysis, goodness of fit, etc., are employed *often and productively* for purposes typically very distinct from those used in their textbook derivation and justification. (For example, multiple regression is typically useful as a generator of residuals and a producer of empirical analytical descriptions and summarizations.)

THE TECHNOLOGY OF DATA ANALYSIS

The basic purpose of a technology is to provide and organize tools and techniques to meet relatively well-specified, but often very broad, objectives. Well-organized technologies are usually associated with better-organized and more basic bodies of knowledge, conveniently referred to as the corre-

sponding sciences. Objectives, science, and technology can only evolve and develop together—and by means of active mutual interaction.

By the term data analysis we mean to encompass the techniques, attitudes, interests and concepts which are relevant to the process of learning from organized records of experience. This area has always been of fundamental importance. It is quite apparent that, currently and in the near future, widespread harnessing of the explosive potential of organized data analysis depends upon active development of its technology. The progress of that development suffers from the fragmentary understanding of both the science and the proper objectives of data analysis. Moreover, it seems to be true that both the mathematical developments of modern statistical theory and the glamor of computer and display hardware have, for different reasons and different persons, provided a diversion from the socially and scientifically important challenges of statistical data analysis. Still, the mushrooming opportunities of modern computing and display provide a major stimulus.

In thinking about data analysis technology, the antithesis between hardware and software (between machinery and organized know-how) is important, not only in the conventional uses of these terms within a computer system, but also in data analysis itself. Specific techniques, such as a three-way analysis of variance, considered as parts of data analysis, are really data-analytic hardware. The mystery and art of when and why to use such techniques make up the data-analytic software, which is today very soft indeed.

Our difficulties are twice compounded; we must develop data-analytic software to harness the power of our data-analytic hardware and, at the same time, develop computer software for data analysis that adequately harnesses the power of our computer and display hardware.

Flexibility in our objectives must be combined with easy and effective iteration and combination. Flexibility, easy iteration, and efficiency all demand the identification and use of functional components that can be assembled in many ways. Existing techniques need to be decomposed into appropriate components; new components need to be recognized and created. Among these functional components we shall find such diverse things as data structures, output formats, algorithms, logical operations, and even components for the construction of other compo-

nents. Their selection and description needs improved guidance from an appropriate appreciation and structuring of objectives. By the same token, their definition and creation will stimulate the evolution of broad objectives and sophisticated techniques.

As far as numerically carrying out classical statistical procedures goes, little has been done, until very recently indeed,²⁰ to attempt to recognize the common arithmetic and logical operations which underlie a great many of the techniques of data analysis, and which may serve as the lowest-level functional components in a more organized approach to it. The recognition of these functional components, and the facility to combine them freely and flexibly, will greatly increase the power and scope of data-analytic methods. Side benefits would also accrue in husbanding programmer effort.

Two considerations are important in implementing data analysis: First, that the process of analysis usually involves a volume of output much greater than the original body of data. Second, that there is no clear barrier between output and input in the overall process of data analysis. The input for analysis is always the output from something else (whether from a previous analysis or a data source). The output from a step of analysis—as for instance an array of residuals or a covariance matrix—is likely to be the input to another phase of analysis. The resulting requirements upon the technology of computation for ease and compatibility are of major importance.

Current facilities for computing, display, and real-time interaction have developed substantially beyond our understanding of how to use them effectively in data analysis. Current limitations in data analysis technology are mainly in explicating and organizing the science of data analysis and in defining and implementing the necessary associated computer software.

From the statistical side of the discipline must come: broader, more permissive, empirically oriented concepts and theories; more inclusive and realistic classifications of objectives; more effective and coherent classifications of useful techniques; research toward more empirically informative techniques that will provide both exposure and summarization; more understanding and research on techniques of reforming and reexpressing variables; deeper insight into the psychology of graphs, pictures and output formats in general, both for inter-

action and for communication; progress toward standardized data structures of great flexibility and comprehensiveness.

From the computing side of the discipline is required software to provide: convenience with flexibility, simple and effective bookkeeping and history keeping, adequate editing, effective means for treating output as input, more flexible and general graphical presentations, and a variety of means to facilitate real-time interaction.

Though some progress is being made on many of these needs, the technology of data analysis is still in its infancy.

ACKNOWLEDGMENTS

We would like to thank G. A. Barnard, D. R. Cox, R. Gnanadesikan, C. L. Mallows, F. Mosteller, H. O. Pollak, and D. R. Wallace for their useful comments on related treatments of the topics considered in this paper.

REFERENCES

1. T. C. Chamberlain, "The Method of Multiple Working Hypotheses," reprint of 1890 version, *Science*, vol. 148, pp. 754-59 (1965).
2. Cuthbert Daniel, "Use of Half-Normal Plots in Interpreting Factorial Two-Level Experiments," *Technometrics*, vol. 1, pp. 311-42 (1959).
3. F. J. Anscombe and J. W. Tukey, "The Examination and Analysis of Residuals," *Technometrics*, vol. 5, pp. 141-60 (1963).
4. G. E. P. Box and D. R. Cox, "An Analysis of Transformations," *Jour. Roy. Stat. Soc.*, vol. 26, pp. 211-43 (1964).
5. Jane F. Munk and M. B. Wilk, "Detecting Outliers in a Two-Way Table," unpublished manuscript (1966).
6. E. S. Pearson, "Some Aspects of the Geometry of Statistics: The Use of Visual Presentation in Understanding the Theory and Application of Mathematical Statistics," *Jour. Roy. Stat. Soc. (A)*, vol. 119, pp. 125-49 (1956).
7. R. Duncan Luce and John W. Tukey, "Simultaneous Conjoint Measurement: A New Type of Fundamental Measurement," *Jour. Math. Psych.* vol. 1, pp. 1-27 (1964).
8. Peter G. Moore and John W. Tukey, "Answer to Query 112," *Biometrika*, vol. 10, pp. 562-68 (1954).
9. J. B. Kruskal, "Analysis of Factorial Experiments by Estimating Monotone Transformations of the Data," *Jour. Roy. Stat. Soc. (B)*, vol. 27, pp. 251-63 (1965).
10. R. N. Shepard, "The Analysis of Proximities: Multidimensional Scaling with an Unknown Distance Function, I," *Psychometrika*, vol. 29, pp. 125-40 (1962).
11. —, "Analysis of Proximities," pt. II, *ibid*, pp. 219-46.
12. —, "Analysis of Proximities as a Technique for the Study of Information Processing in Man," *Human Factors*, vol. 5, pp. 33-48 (1963).
13. J. B. Kruskal, "Multidimensional Scaling by Optimizing Goodness of Fit to a Non-Metric Hypothesis," *Psychometrika*, vol. 29, pp. 1-27 (1964).
14. —, "Non-Metric Multidimensional Scaling: A Numerical Method," *ibid*, pp. 115-29.
15. M. A. Efroymson, "Multiple Regression Analysis," in *Mathematical Models for Digital Computers* (Anthony Ralston and Herbert S. Wilf, eds.), Wiley and Sons, New York, 1960, pp. 191-203.
16. Robert G. Miller, "Statistical Prediction by Discriminant Analysis," *Meteorological Monographs* (Boston, American Meteorological Society), vol. 4, no. 25 (1962).
17. Norman J. MacDonald and Fred Ward, "The Prediction of Geomagnetic Disturbance Indices: 1. The Elimination of Internally Predictable Variations," *J. Geophys. Res.*, vol. 68, pp. 3351-73 (1963).
18. M. B. Wilk and R. Gnanadesikan, "Probability Plotting Methods for the Analysis of Data," unpublished manuscript (1966).
19. John W. Tukey, "The Future of Processes of Data Analysis," *Proceedings of the 10th Conference on the Design of Experiments in Army Research, Development and Testing*, U. S. Army Research Office (Durham), 1965, pp. 691-729.
20. Albert G. Beaton, "The Use of Special Matrix Operations in Statistical Calculus," Ed.D. thesis, Grad. School of Education, Harvard University 1964.

UNICON COMPUTER MASS MEMORY SYSTEM

C. H. Becker

*Precision Instrument Company
Palo Alto, California*

PRINCIPLES

The principle of the UNICON Computer Mass Memory is derived from the UNICON Coherent Light Data Processing System to create and detect (record and reproduce) information elements in two dimensions by means of signal-modulated coherent laser radiation. The UNICON Computer Mass Memory System has the following characteristics.

An information bit is represented by a diffraction-limited "hole" within the Unidensity film layer. It results from the evaporation of the Unidensity medium by means of imaging the aperture of a laser to a three-dimensional ellipsoid of revolution (Debye ellipsoid). During the time required to create one bit, the vacuum temperature of the laser image of 3×10^4 °K produces a vapor pressure of 1000 atmospheres within the bit volume, leaving an ellipsoidal hole in the Unidensity layer of the following dimensions (see Fig. 1).

Large axis ($2Z_0$) in the direction of propagation of the writing laser beam:

$$2Z_0 = 4 \times \lambda \left(\frac{f}{D} \right)^2 \quad (1)$$

Small axis ($2r_0$) transverse to the direction of propagation of the writing laser beam:

$$2r_0 = 2 \times 1.22 \lambda \left(\frac{f}{D} \right) \quad (2)$$

It is assumed that the thickness t of the Unidensity layer is approximately equal to the length of the large axis $2Z_0$.

Transfer of the information to be stored to the laser beam utilizes electro-optical modulation of the beam. Primarily, information is presented to the modulator as frequency modulation or pulsewidth modulation.

Instantaneous readout of the three-dimensional information bit, while recording, occurs by detecting the diffracted laser radiation during evaporation of the Unidensity medium. Without information, the logarithm of transmissivity T of the Unidensity layer is inversely proportional to its optical density s :

$$s = \log \frac{1}{T} \quad (3)$$

After evaporation, this density is practically reduced to zero, thereby increasing the power of the laser beam transmitted through the bit area by approximately 70 decibels. Secondary readout of the information takes place at drastically reduced laser power to avoid further destruction of the Unidensity film layer.

Continuous readout of the UNICON system utilizes a lightguide surrounding the imaging circle of the rotating objective, carrying the laser radiation transmitted through the Unidensity film to a central photomultiplier. Hence, any coherently illuminated

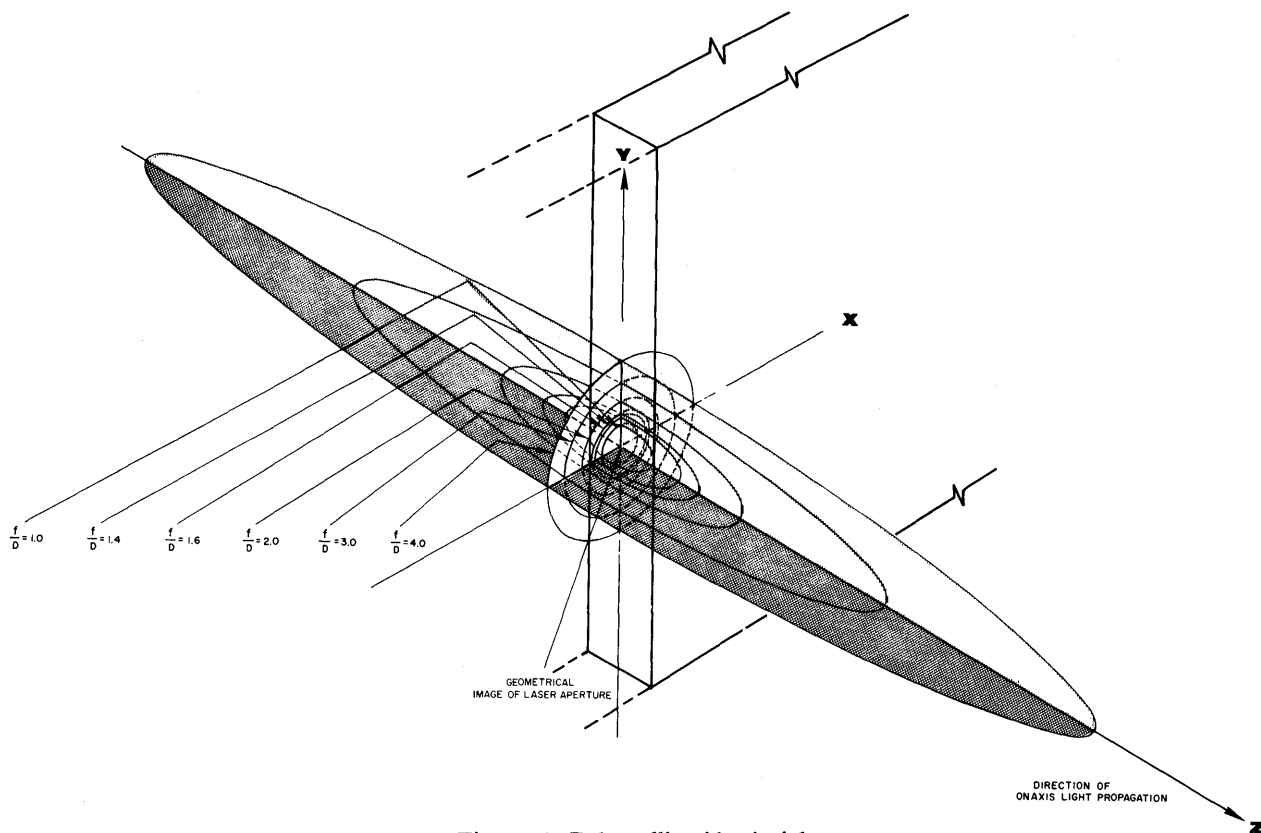


Figure. 1. Debye ellipsoid principles.

information bit is photoelectrically detected within a few nanoseconds. This holds true for instantaneous and secondary readout as well.

In order to establish a two-dimensional pattern of information bits, the Unidensity layer is helically carried at slow speed (1 cm per second) around the imaging circle of the laser aperture, which is formed by a diffraction-limited objective rotating with high speed around the helical axis. The velocity of the laser image is 18.6 meters per second at 1800 rpm of the rotating optical system. The inclination of the image azimuth against the transport direction of the Unidensity film is $1^{\circ}32$ minutes. Hence, helical unit records of one line length are recorded at 600,000 bits each, with spacing between individual unit records of 2 bit diameters. Width of the information-carrying area of the 16mm Unidensity film is 8 mm. Information packing density is 6.45×10^8 bits per square inch. Rate of information processing is in the megabits-per-second range. Total capacity of one UNICON Memory System is 88×10^9 bits for a 16mm Unidensity film reel of 100 feet.

Compared to the present state of the art, which is characterized by memory cards, discs and drums, the

helical two-dimensional UNICON memory represents an entirely new system with several orders of magnitude higher performance.

This advancement in the state of the art is due essentially to the basic principle of the UNICON system, that information storage takes place by moving the entire storage medium across the ultimately precise imaging circle of the system, continuously and without spatial intermittence. Hence, the UNICON Computer Mass Memory is indeed a two-dimensional memory system which does not require stepping processing in space to move cards, to change circular writing and reading positions on discs, or to vary height positions on drums. In other words, the UNICON system provides an imaging circle for recording and reading of practically invariant radius and fixed positions in space. The information guidance problem of the UNICON system is therefore reduced to the kinematics of the continuous helical motion of the information carrier across the imaging circle. Due to the unique design principles of the UNICON system, this guidance is also practically invariant in space and is inseparably connected to the spatial characteristics of the imaging circle of the system.

COHERENT LIGHT BIT CREATION

Bit creation in the UNICON Computer Mass Memory takes place by means of evaporating a diffraction-limited hole in the special Unidensity medium, utilizing coherent laser radiation as the writing power source. In order to accomplish this operation, certain basic requirements must be met:

1. The information-writing laser beam must be of zero-order, single-mode (TEM₀₀) structure; otherwise, the laser image will not be a single ellipsoid of revolution and of the smallest possible size.

2. The power density of the writing laser beam must be such that a certain temperature of vaporization (3×10^4 °K) is established within the ellipsoidal volume of the laser image.

3. The information storage medium (Unidensity layer) must possess such absorption and reflection characteristics that a certain vapor pressure (1000 atmospheres) is established in the ellipsoidal volume of the laser image under the influence of the beam temperature in this volume.

4. The imaging optics which concentrate the writing laser beam must be free of optical distortion within the diffraction limits and must possess certain aperture ratio (f/D) of the focal length f and effective aperture D so that the dimensions of the three-dimensional bit volume (ellipsoid of revolution with large axis $2Z_0$, and small axis $2r_0$) are of the smallest possible size (Fig. 1).

$$2Z_0 = 4 \times \lambda \left(\frac{f}{D} \right)^2 \quad (4)$$

$$2r_0 = 2 \times 1.22 \lambda \left(\frac{f}{D} \right) \quad (5)$$

where λ is the laser wavelength.

5. The Unidensity storage medium must be of such thickness, measured in the direction of propagation of the writing laser beam, that the ellipsoidal volume is properly embedded.

6. The transverse position of the laser Unidensity film must be so accurate that secondary tracking from the reading laser beam can be accomplished within the track width of a helical unit record. (Track widening of the reading laser image to a slit transverse to the scanning direction relaxes these requirements, for example, by a factor of 3.) A track widening slit, transverse to the scanning direction, appreciably decreases the problem of tracking the

reading laser image within the track width of a helical unit record. The reading laser image is increased by a factor of three through this process.

7. The longitudinal positioning of the writing and reading laser beam must be sufficiently accurate that azimuth tracking takes place within the track width of the helical unit record. Due to the very small slope of the unit record track ($1^\circ 32'$), this requirement can be met by servo control of the Unidensity film transport.

Under these conditions, the bit creation of the UNICON system takes place as follows: The intrinsic aperture D of the writing laser beam, with its characteristic divergence α (see Fig. 2) is imaged by the diffraction limited objective

$$\sin \alpha = 1.22 \left(\frac{\lambda}{D} \right) \quad (6)$$

Due to the diffraction phenomena, the image of the laser aperture is not a geometrical point, but an ellipsoid of revolution (Fig. 1).

From the various possibilities to design the imaging structure of the UNICON system, laser beam cross section and divergence are to be matched to the numerical aperture of the imaging objective and its focal length in accordance with the Unidensity layer thickness t and the required bit size $2r_0$.

Assuming the imaging objective is completely "filled" at a certain distance from the laser exit (i.e., laser beam diameter equals objective entrance pupil), another concentration of the power density S occurs, proportional to the square ratio of laser beam diameter and spot diameter:

$$\frac{D^2}{\left[2 \times 1.22 \lambda \left(\frac{f}{D} \right) \right]^2} \cos^2 \Theta_0 \quad (7)$$

where Θ_0 is the space angle of laser imaging.

One obtains, for example, with $W = .350$ watt, $2r_0 = 1$ micrometer, and $S = 8 \times 10^{10}$ watts/meter².

Applying Stefan Boltzmann's law to this power density S

$$S = \sigma T^4 \quad (8)$$

where

$$\sigma = 5.673 \times 10^{-12} \text{ watt cm}^{-2} \text{ }^\circ\text{K} \quad (9)$$

one obtains the resulting vacuum temperature T to be

$$T = 3 \times 10^4 \text{ }^\circ\text{K} \quad (10)$$

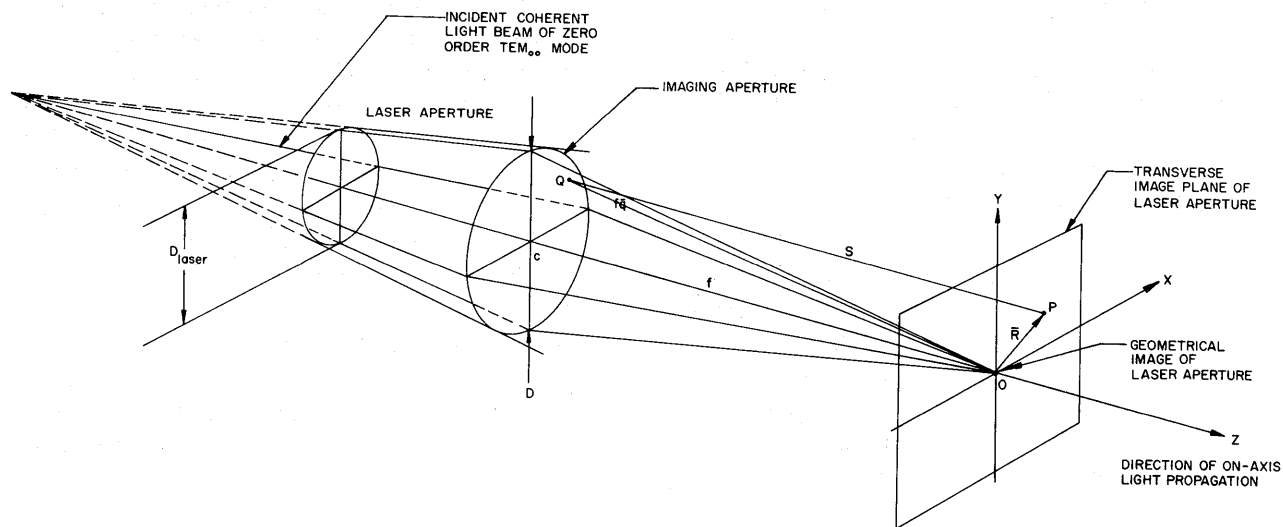


Figure 2. Optical rays for theory of imaging of a laser aperture (principles).

Applying further the Clausius Clapeyron equation to the Unidensity layer and this temperature:

$$\frac{dp}{dT} = \frac{U}{T\Delta V} \quad (11)$$

where p = vaporization pressure,
 U = latent heat of evaporation,
 T = absolute temperature, and
 ΔV = molar specific volume of the evaporative Unidensity medium,

one obtains, after integration:

$$p = - \left(\frac{V}{Uv\alpha p} \right) S \quad (12)$$

and, with the evaporation heat per unit mass m and the thermal velocity V of the evaporated atom:

$$V = \left(\frac{2}{m} \cdot KT \right)^{\frac{1}{2}} \quad (13)$$

where K is Boltzmann's constant.

Introducing the various parameters related to bit creation, the vaporization pressure is of the order of 1000 atmospheres.

COHERENT LIGHT BIT DETECTION

In principle, coherent light bit detection may be defined as the coherent light illumination of the created information bits during and after creation,

and the subsequent photoelectric detection of the transmitted light diffraction pattern. Optimal conditions require laser and imaging system to be equal for storage and readout, except for drastic reduction of the coherent beam power for detection to avoid erasing.

Hence, with these requirements in mind, one defines the optics for light guidance from information bits to photoelectric detector. In the UNICON system, light transmission to the photomultiplier takes place by means of a plexiglass guide completely surrounding the imaging circle of the laser aperture and guiding the light transmitted through the Unidensity film to a central photomultiplier.

ACCOMPLISHMENTS

Possible mass memory configurations are the result of practical accomplishments and experiments with the prototype UNICON-6. This research program has accomplished the following goals:

1. Determination and definition of the physics involved.
2. Experimental recording of 0.7μ spots and recovery of the data.
3. Construction of a working model.

This working prototype has read-write capability, instantaneous playback without processing, and is of archival quality. These accomplishments and experimental prototype characteristics allow for a reason-

able projection of possible UNICON Mass Memory Systems.

CHARACTERISTICS OF UNICON MASS MEMORY FOR TYPICAL COMPUTER INSTALLATION

The previous discussion described the UNICON-6 prototype Mass Memory System. Let us examine for a moment several typical UNICON Mass Memory configurations that might be used in conjunction with a large computer installation.

While the UNICON-6 uses a 16mm wide recording medium, a 35mm width of recording medium would be more practical, from the standpoint of reducing access time. The track layout would be as illustrated in Fig. 3. The 35mm width in this figure includes 4 mm in the upper edge track for a binary file accession number, 1 mm in the lower edge track for a time track and a 30 mm wide track for recording of data. Data tracks 1 mm in length would be laid transversely along this width, each track providing a file length of 10^6 bits. The track separation normal to the direction of recording would be 4 microns. Due to the low incidence angle, we then have a 167 micrometer track separation distance along the length of the recording medium. This configuration would provide a total information capacity of 10^{12} bits in 528 lineal feet of recording information:

$$\frac{10^6 \text{ bits/record}}{167 \mu \text{ separation} \times 100 \text{ cm}} = 60 \text{ tracks/cm} \tag{14}$$

$$\begin{aligned} & \frac{10^{12} \text{ total memory size}}{10^6 \text{ bits/file} \times 60 \text{ tracks/cm} \times 30.5} \\ & = 528 \text{ lineal feet of recording medium} \\ & \text{for } 10^{12} \text{ bits} \end{aligned} \tag{15}$$

A desirable configuration to minimize access time would be to provide one-half of the recording medium on each of two reels, with the transport controls set to automatically home in the center position. Since our average access time would be the time required to access a file half the distance in length on a roll of recording medium, we can then calculate the average access time as follows:

$$\begin{aligned} \text{Search velocity} &= 120 \text{ ips} \tag{16} \\ \text{Average access} &= \frac{528 \times 12}{120} \times \frac{1}{2} \times \frac{1}{2} = 13.1 \text{ sec} \tag{17} \end{aligned}$$

This average access time is then 13.1 seconds for the entire 10^{12} -bit memory. The worst-case access time would, of course, be double this figure, or 26.2 seconds.

A configuration suitable for an application such as a library retrieval system accessed by many on-line users through a multiprocessing computer might be as shown in Fig. 4. Note that in this case, 10 transports are provided, each having a capacity of 10^{11} bits. In this configuration our average access time would be 1/10 of the former case, or 1.31 seconds, neglecting start/stop time. Each transport is equipped with a transport controller which will receive the address of a desired file from the master

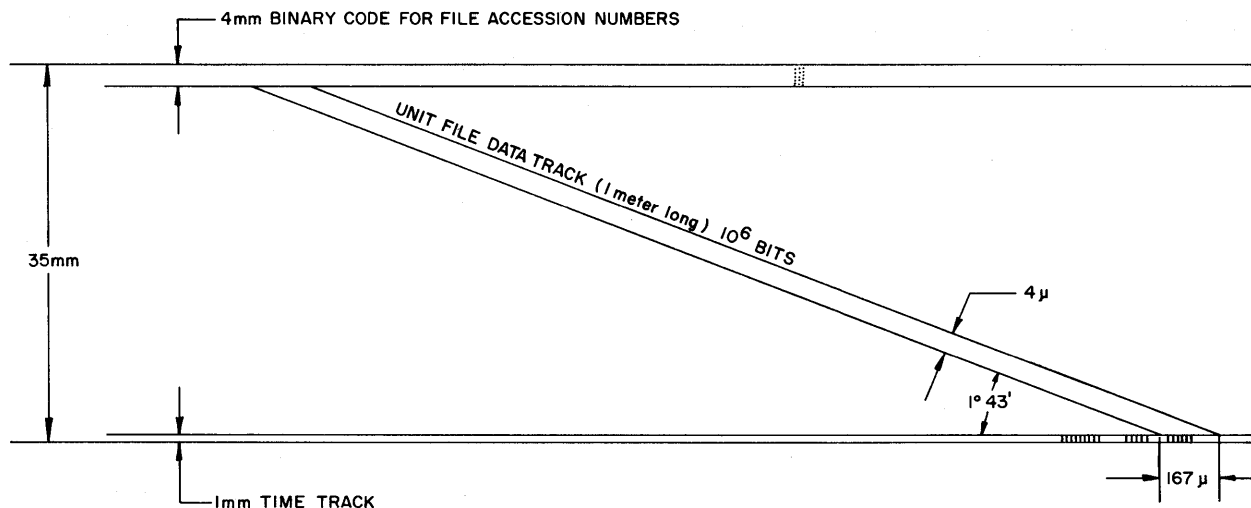


Figure 3. Track layout.

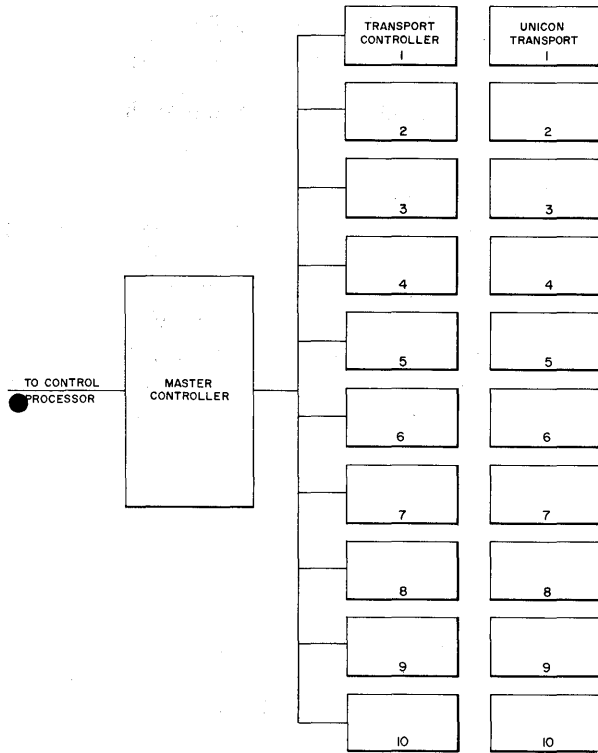


Figure 4. Typical large memory system configuration.

controller and proceed to search for the file. Upon locating the desired file, the transport control will interrupt the master control and dump the entire contents of the selected file into the master controller buffer.

The master controller would have the capability of receiving multiple commands from the computer and call for files on a first-in first-out basis. By incorporating a small processor within the master controller, it would be perfectly feasible to issue file requests to individual transports in such a manner as

to minimize total access time. This capability would require the master controller to have knowledge at all times as to the physical position of the recording medium in each transport and issue those commands to each tape transport which could be accessed with minimum linear travel from its present position. This type of operation would, of course, require some users to wait longer than others to reach a particular file, but would have the advantage of maximizing memory system output.

Data Transfer Rate

The data transfer rate can be varied up to the megabit rate to accommodate the word length and data transfer rate of the highest speed core memory systems. This is accomplished by adjusting the rotational speed of the optical scanner head to scan the data track at the appropriate rate.

The above two configurations of mass memory systems show two possible configurations of the UNICON Mass Memory. With appropriate controllers, it is possible to configure the UNICON Mass Memory to accommodate most multiprocessing and multiprogrammed computers.

ACKNOWLEDGMENTS

The author wishes to acknowledge the cooperation of all who contributed to the establishment of the UNICON Computer Mass Memory at Precision Instrument Company, particularly Messrs. Siegfried Mohr and Herman Wong. Acknowledgment is also given to William Bennett, Vice President—Marketing of Precision Instrument, who contributed the section of this work on “Computer Characteristics of UNICON Mass Memory.”

AN ELECTRON OPTICAL TECHNIQUE FOR LARGE-CAPACITY RANDOM-ACCESS MEMORIES

Sterling P. Newberry

*General Electric Company
Schenectady, New York*

INTRODUCTION

Memories of the electron beam recording type have many desirable features for large capacity applications. At the Wescon Conference of 1958,¹ the author proposed a class of electron optical memories of very high storage density under the title, "Information Storage in Microspace."

The intent of the microspace approach was to increase the storage density to a level at which the entire memory surface could be so small that:

1. The memory could be enclosed in a convenient-sized, controlled environment, such as a vacuum chamber, free from dust, stray fields, temperature excursions, and other extraneous influences.
2. The memory surface would not be liable to injury from being rolled or folded upon itself.
3. The mechanical motions could be drastically reduced or eliminated altogether.
4. The difficulty of relocation of the data, which increases monotonically as the memory capacity is increased, could be lessened by organizing the memory in segments with servo-control of the

electron beam to and within a given segment.

The desirability of completely eliminating mechanical motion so that rapid random access could be obtained to any part of the memory soon became apparent. However, this goal is unattainable with a single electron lens because of the limitation in field of view of a single lens. One suggestion for getting around this limitation, and thus for increasing the amount of storage surface which can be in focus simultaneously, was made by the author at the Fourth Electron Beam Symposium in 1962.² This suggestion was for the creation of a matrix of electron lenses resembling in principle the compound eye of the insect world and thus called a fly's eye lens. In this matrix of lenses, which may be either an electrostatic or electromagnetic array, each lenslet will be capable of keeping that part of the memory surface immediately before it in focus at all times, ready for instant recording or readout.

TYPICAL FLY'S EYE STRUCTURE

A schematic diagram of a cross section of such a device employing electrostatic lenses of the "Einzel" type is shown in Fig. 1. Here the lenslets are arranged in a rectangular array. The inset, at lower right of figure, shows one lenslet of the complete

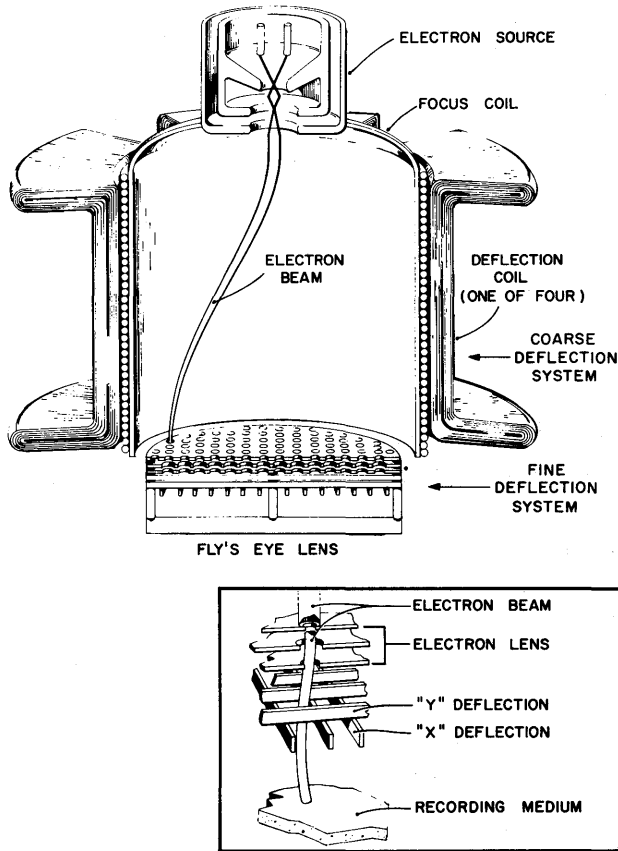


Figure 1. Schematic diagram of microspace concept employing fly's eye lens.

lens in greater detail. It may be seen that the lens is an array of simple Einzel lenses formed as usual by three apertures on a common axis.

All the center apertures of the lenslets are contained in a common metallic plane sheet which is maintained at a potential approaching cathode potential, or at least negative with respect to the electron beam potential. In like manner the outer apertures of all the lenslets are contained in a common metallic plane, for each side respectively, and these outer planes of apertures are at anode potential, which is customarily also the ground potential of the system. Thus for the complete matrix of lenslets three leads are required, one for each plane of apertures. It does not matter that all lenslets are connected, since only that lenslet or lenslets to which an electron beam is directed will be active. Thus if a common electron source is employed with a coarse deflection system as shown in Fig. 1, it can be used as an electrical switch to activate any required lenslet on command.

Immediately following each lenslet a set of X and Y deflection plates, forming a fine deflection system, is shown. From the inset it may be seen that the deflection plates for each row of lenslets form a continuous deflection bar. The other set of deflection plates form a continuous set of bars for the columns of lenslets. Thus, as in the case of the lens plates, a few connecting leads serve to supply voltage to all of the lenslets' deflection plates since it does not matter that a deflection field exists in every lenslet. Only the one lenslet to which the electron beam is addressed is activated. The direction of deflection in adjacent lenslets is reversed but this is of no special consequence for most possible applications.

This fine deflection system is then followed by the recording medium as shown. The precise form of the recording plate depends upon the properties of the medium, which is beyond the scope of the present paper. For purposes of the tests reported here Lippmann-type photographic emulsions have been employed. The above straightforward arrangement has been described in detail because it is the form which has received the most extensive testing and upon which the results reported here were performed. By way of example, a possible variation might contain the focus and deflection functions in the same structures, or the lenslets might be in hexagonal array instead of rectilinear array as shown.

CONSTRUCTION OF A TEST MODEL

As a first test model, it was decided to build a 10 by 10 lens matrix on $\frac{1}{16}$ " centers (which is roughly $1\frac{1}{2}$ millimeters). The complete matrix of 100 lenses was therefore contained in an area of 2.25 square centimeters although the supporting structures extended beyond the lens matrix to a diameter of approximately 5 cm. The central aperture was chosen to be 0.010" (0.25 mm) diameter and the spacing between planes the same value. The deflection bars were made by simply milling slots in metal plates to form four comb-like structures which could be interdigitated to give the required two sets of deflection plates. A completed assembly is shown in Fig. 2.

EXPERIMENTAL SETUP FOR TEST

The test of the fly's eye unit was conducted in a demountable electron optical bench similar to the bench described by Ruska.³ The test arrangement is



Figure 2. Completed assembly of 10 by 10 matrix fly's eye lens.

shown schematically in Fig. 3. Only the top section of the equipment is illustrated. The electron source and condenser lens assembly (not shown) were respectively a standard hairpin filament source and an electrostatic condenser lens described previously by the author.⁴ This source size has been found to be approximately 75 microns in diameter. The image and object distances were set to give a demagnification of 60 times. A fine-grain (vapor reacted) phos-

phor screen* shown in Fig. 3 is placed at the focal position of the fly's eye structure seen immediately below the fluorescent screen. Above the screen is a light microscope objective contained in the vacuum. The objective can be focused from outside by means of the bevel gear train. The light microscope viewing system is completed by an eyepiece external to the

* Vapor reacted screen obtained from Liberty Mirror Division of Libbey-Owens-Ford.

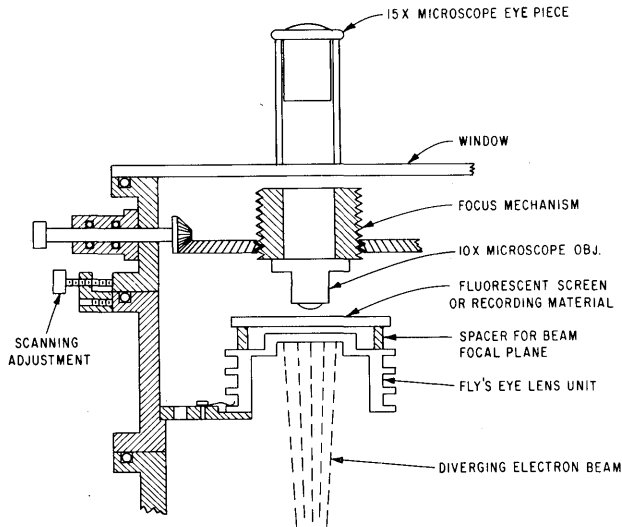


Figure 3. Schematic of test arrangement for fly's eye lens evaluation.

vacuum window. Viewing magnification could be varied from $50\times$ to $400\times$ by exchanging objectives and eyepieces.

A photograph of the test setup, on the electron optical bench, is given in Fig. 4. In the vertical column of electron optical components, the electrons proceed from bottom to top. The column is 4" in diameter (approximately 100 mm) and is connected to the vacuum system by a metal cone seen to the lower right of the column. This cone is welded to one section of the column and serves also as the mechanical support for the column. Just below the cone may be seen the insulator for the electron source described in Ref. 4. The section immediately above the cone houses the electrostatic condenser lens of Ref. 4. Centering of the elements is accomplished by sliding action of the "O" ring seals between sections and is controlled by means of the external collars and thumb screws which may be seen in the photograph at the junction of some adjacent sections. The next three short sections house the fly's eye lens. The top of these three sections is made of lucite plastic to aid in beam current measurement and introduction of the focus voltage to the lens. The middle section contains six insulators, four of which are used to introduce the deflection voltages, one for providing ground potential to the top lens plate and the last not used since the bottom lens plate is fastened to the lowest of the three sections and thus provided with adequate ground connection and mechanical support. The fluorescent

screen or photographic plate is held at the correct focal plane for the fly's eye lens by a simple annular, ceramic, ring spacer placed to rest on the outer edge of the lens. The fluorescent screen has a conducting coating and is connected through the plastic section to a lead for either simple ground connection or connection to a beam current meter with appropriate positive bias to collect secondary electrons and obtain a true reading. The photographic plate is exchanged for the fluorescent screen by breaking vacuum and lifting off the top two sections. For light sensitive materials this operation is conducted with the aid of a red cellophane filter in a flashlight. The photographic plate is kept from moving during exposure by a large-diameter, weak coil spring pressing down on it. Obviously, the system must maintain

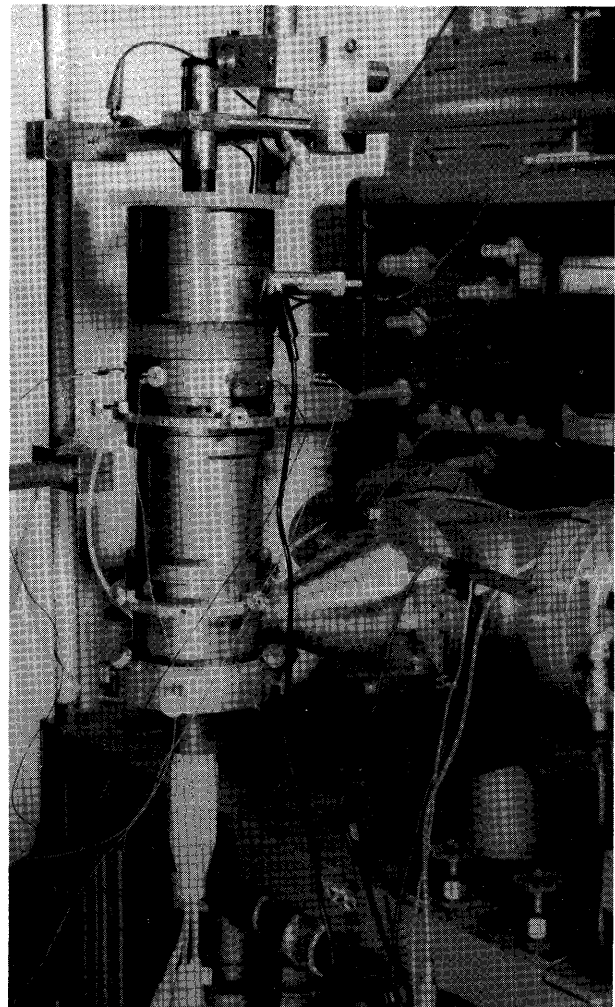


Figure 4. Photograph of test arrangement for fly's eye lens evaluation on electron optical bench.

alignment and focus through this vacuum cycle. The top two sections contain the objective lens assembly of the viewing microscope with its planetary gear system for external focus control obtained by the knob seen on the right. The top section allows space for this focus motion. The eyepiece of the viewing microscope is external to the vacuum system and may be seen at the top of the column. This arrangement places the vacuum chamber window between the objective and the eyepiece where it introduces negligible aberration. To the left of the column one may see the polyethylene high voltage leads which supply condenser lens and fly's eye lens focus voltages. Their respective high-voltage plastic bushings are on the back side of this view and therefore cannot be seen. These bushings are completely enclosed for safety reasons, as are the leads to the electron source seen at the bottom of the column.

TEST OF THE 10 BY 10 MATRIX

For the first test, the light microscope was removed after focusing and the electron source was made to flood all of the lenslets with a 4-keV beam by causing the beam to cross over close to the condenser lens. The simultaneous focusing action of the lenslets with grounded deflection plates was observed on the fluorescent screen. The fluorescent screen image was recorded by 35mm photography, as shown in the image sequence in Fig. 5. Subsequently, the light microscope was reinstalled and the fine deflection system of the fly's eye was connected to the plates of a type 536 Tektronix oscilloscope, which gave a maximum potential of 160 volts in one direction and 80 volts in the other. The plates also contained a DC bias of approximately 80 volts. A simple sawtooth pattern was observed on the fluorescent screen by use of a 16-mm 0.25 NA objective. A photographic plate was then used in place of the fluorescent screen by opening the vacuum system and then exposing the plate without the possibility of reexamining the focus. The sawtooth trace approached 1.5 microns at the narrowest part which is consistent with a demagnification of about 60:1 and a source size of about 75 microns and gave encouragement to attempt image recording through the lens.

IMPROVED TEST SETUP

Before making test recordings the setup shown in Fig. 4 was added to and improved in the following

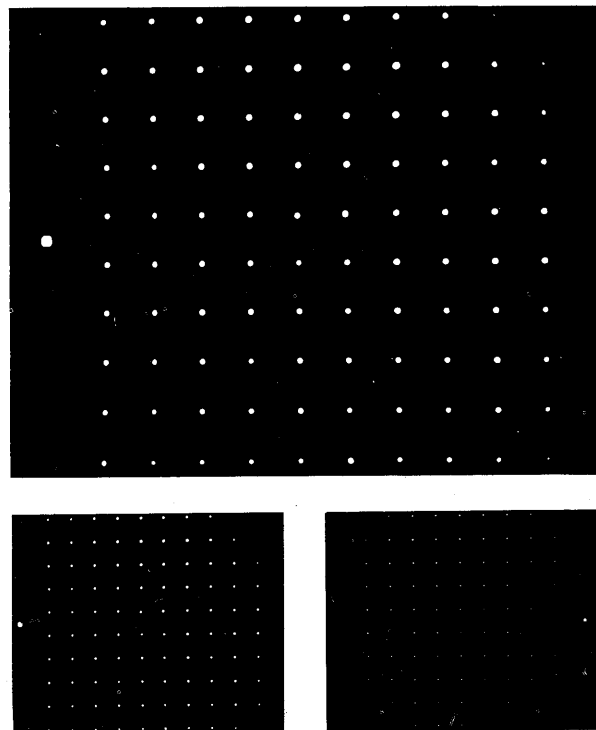


Figure 5. Focal sequence of 10 by 10 fly's eye lens. *Top:* .625 × .625 fly's eye electron beam pattern; beam entered corner lenslets at an angle. *Bottom:* Two stages of focus through all lenslets.

manner. A stator type television focus coil* was added between the condenser lens and the fly's eye unit using a brass tube through the center of the coil to keep it outside the vacuum chamber. The brass tube had flanges connected to each end by threaded joint and "O" ring seal to make it compatible with the rest of the sections in the electron optical bench column. While this deflection unit permitted the use of areas of the fly's eye away from center, it could not adequately direct the beam to the outermost lenslets because the angle of the beam to the lens normal increased with deflection. In the latest version, described below, this limitation is removed by employing double deflection so that a second set of coils straightens the beam back to the lens normal just before it reaches the required lenslet opening. The simple deflection control from the 536 oscilloscope was replaced by control from an image orthicon chain.† Signals are fed, through appropriate video amplifiers, from the chain to the fine deflection bars and the grid of the electron source. Deflection

* Celco Type AY 521-5600 (Constantine Engineering Laboratories Co.).

† General Electric Model URV-200.

levels available were ± 400 volts and grid swing was up to -20 volts coupled to the gun with a .02 Mfd, 6 kv capacitor. After the first images of a test chart, shown in Fig. 6, were recorded the fly's eye unit was mounted in a more convenient electron optical bench section designed specifically for holding it, and a plate holder was added with a linear motion feed-through so that the fluorescent screen and photographic plate could be interchanged over the fly's eye unit without breaking vacuum. It was still necessary to break vacuum to place the photographic plate into or out of the system however. An adjustable aperture similar to the one described in Ref. 4 was installed between the source and condenser lens and a simple mechanical shutter with a Faraday cup on its extremity was installed to control exposure and measure total beam current. Finally the hairpin fila-

ment of the electron source was given a small pointed end after the method of Hibi⁵ to decrease the source size without loss of brightness. For visual focusing the current at the fluorescent screen was raised to 10^{-7} amperes or higher, but for photography the current had to be reduced below 10^{-9} amperes to give a convenient time of 1 to 10 seconds for mechanical exposure. It is not possible to determine by fluorescent screen viewing whether the resolution suffers at higher beam current but no deterioration is expected. This item will be tested when single trace photographic control becomes available.

PHOTOGRAPHIC RESULTS

Photographic emulsions of Lippmann-type were used to test the performance of the fly's eye lens. At

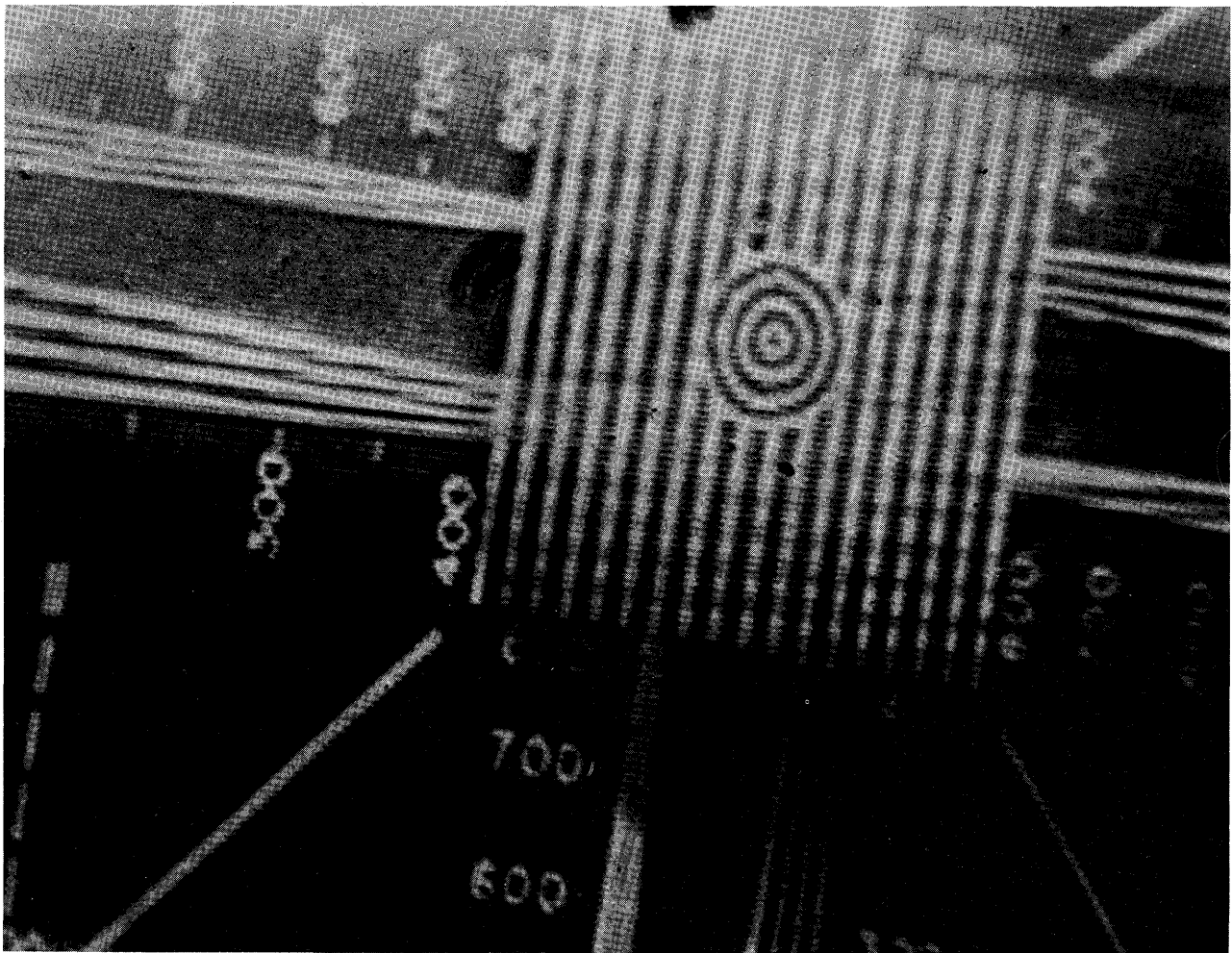


Figure 6. Portion of RETMA resolution chart by scanning action of one lenslet recorded on photographic film. Magnification marker—100 microns.

first Kodak high-resolution plates were used but the 4-kV beam energy was insufficient to reach the silver halide grains so that recordings were essentially due to changes made in the surface of the gelatin. This difficulty was overcome by making emulsion coatings of low gelatin content according to the method described by Salpeter and Bachmann.⁶ Later on Eastman Kodak produced some experimental Lippmann-type emulsion of low gelatin content and grain diameters around $50 \text{ m}\mu$, which was more convenient to use. A good description of Lippmann emulsion is given by Mees and James.⁷ This emulsion can be spread in thin layers after the method of Hamilton and Brady⁸ and mounted on a glass slide or electron microscope specimen grid. It has been

supplied to us through courtesy of the Kodak Company on a purely experimental basis and no assurance can be given regarding future availability. The pictures shown here were made with this emulsion on tin-oxide-coated microscope slides. Development was for one minute in D-19 developer. Hamilton and Brady suggest a developer containing ascorbic acid for best resolution, but it is less stable and not required at the present resolution level.

The image shown in the photomicrograph of Fig. 6 was produced by scanning action by one lenslet of the 10 by 10 matrix of lenslets. The image is of the RETMA resolution chart #1956, the photomicrograph was made with a $10\times$ objective. Figure 7 is from the same recording as Fig. 6 but taken with a

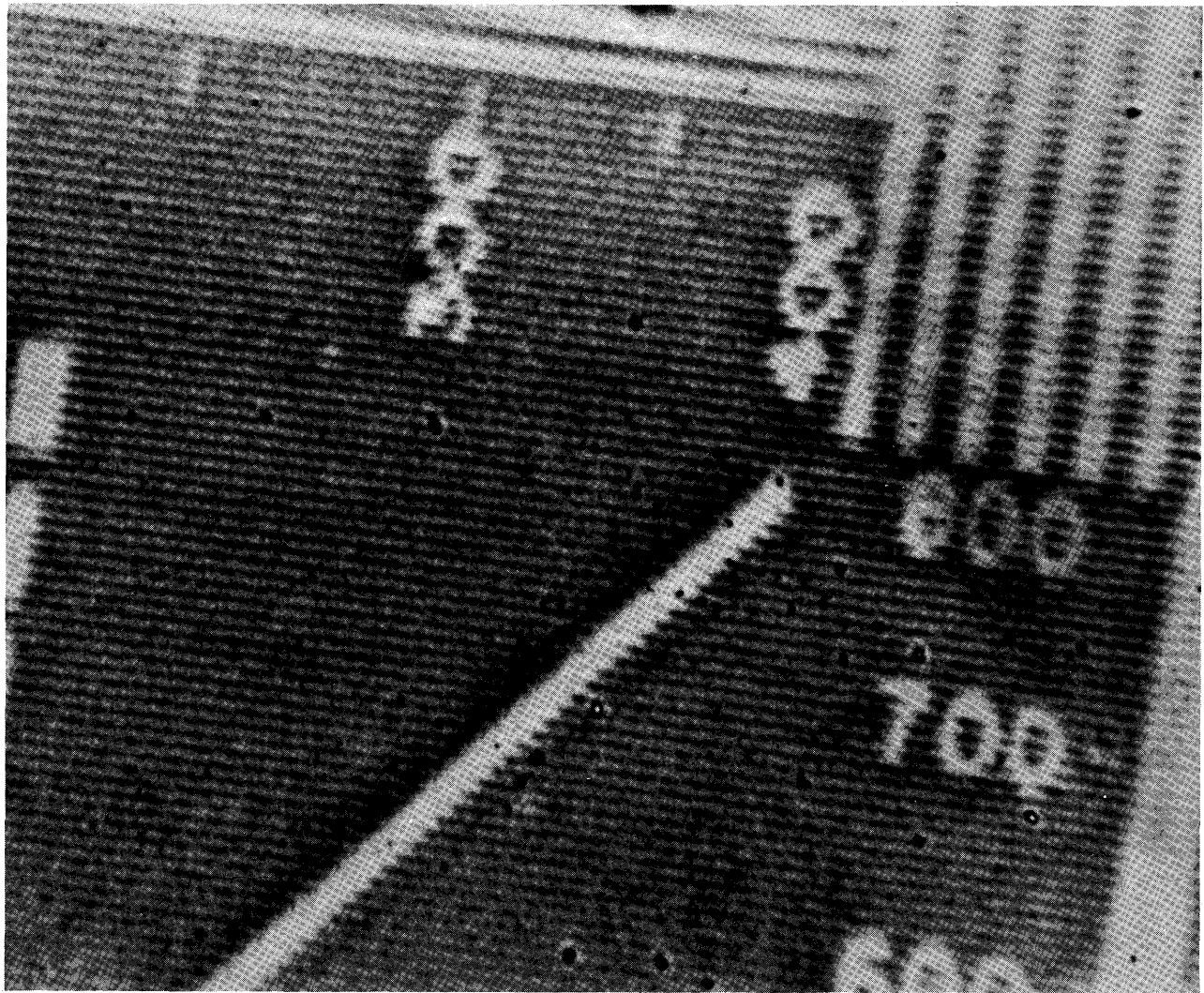


Figure 7. Higher-resolution microphotograph of recording shown in Fig. 6. Magnification marker—10 microns.

20 \times objective. For a restricted field of view, it shows the individual scan lines in better detail. Even Fig. 6 does not cover the entire picture. It is unfortunate that the optical microscope cannot both resolve the scan lines and cover the field of view of the entire recording although it may display either one by itself when the proper objective is used. At the border of Fig. 7 is a magnification scale representing 10-micron spacing between centers of adjacent lines. This scale was produced by photographing a 10-micron Bausch & Lomb scale immediately after photographing the fly's eye image and with the same setting of the microscope except that dark field illumination was used to make the scale lines more distinct but, of course, reversing them from black to white. The Leitz ortholux microscope and 35mm orthomat camera were used to take the pictures. In both Figs. 6 and 7 the path of the electron beam on the printed page is black. We have a further check of their identity because the test chart bars and numbers appeared black on the fluorescent screen prior to recording and therefore must be opposite to the electron beam, which glows brightly on the fluorescent screen.

For these pictures electron optical demagnification was approximately 20:1, the object and image distances being 10" and 0.5" respectively. Assuming negligible lens aberration a source size of 20 microns is indicated which is reasonable. Beam currents of up to 1 microampere were observed on the fluorescent screen but beam current had to be reduced to 5×10^{-10} amperes to give a reasonable exposure time for the Kodak experimental emulsions and mechanical beam shutter used. Again, the beam voltage was 4 kV and the deflection voltage was ± 400 V to give a deflection of $\frac{1}{16}$ " to match the lens matrix spacing.

PRODUCTION OF A 32 BY 32 MATRIX LENS

After the initial test of the 10 by 10 matrix lens and during the time of the later tests, an improved version of the fly's eye structure was produced as seen in Fig. 8. Its chief differences were improved tolerances in centering the plates during assembly, production of deflection bars which were anchored at both ends and a 10-fold increase in the number of lenslets to a 32 by 32 matrix on $\frac{1}{32}$ " centers (approximately $\frac{3}{4}$ mm). This lens has given the improved pictures shown in Figs. 9 and 10, which contain lines less than 1 micron on two micron

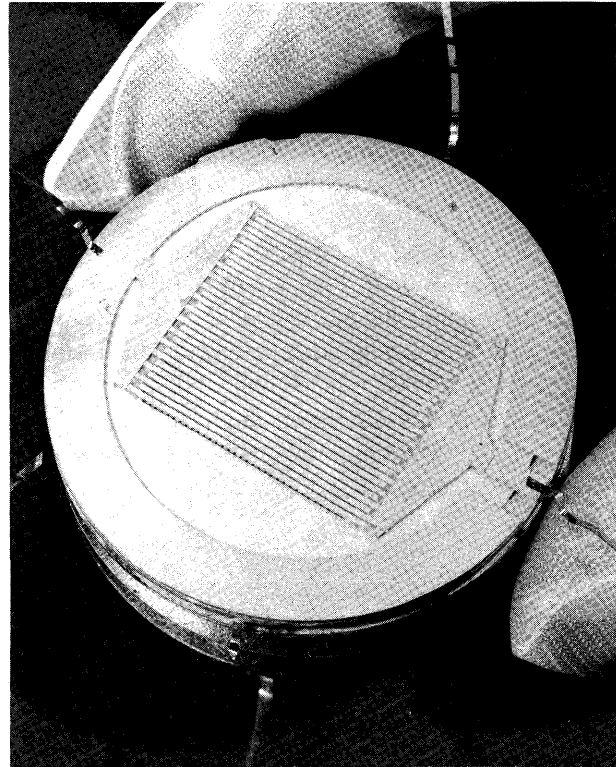


Figure 8. Completed assembly of 32 by 32 matrix fly's eye lens.

centers. Figure 9 shows simulated digital data being scanned by four neighboring lenses simultaneously. The apparent lack of linearity is chiefly due to the signal source. This becomes apparent when it is recalled that adjacent lenses give mirror images. The distortions are seen to be reproduced identically by each lens. Figure 10 is included to give some idea of gray scale response and shortness of exposure. (Figure 10 was taken "live" not from a photograph. Also, an extra photographic reversal was introduced to avoid a negative for the final print.) The improved test equipment shown in Fig. 11 has been provided and a small electron microscope has been constructed to permit focusing the electron beam at submicron dimensions. For these proposed tests the demagnification may be increased to 50:1. If the lenses were mechanically perfect, the aberration limit would be expected to be 0.03 microns. Photographic film has been shown by Salpeter and Bachmann⁶ to be capable of an average grain size of 0.05 microns. Thus a resolution of 4000 line pairs per millimeter is a worthwhile goal to strive for, since we have emulsion capable of recording at this level.

IMPLICATION OF RESULTS

The full resolution of the available 480 lines of the television signal indicates better than 2×10^6 clearly resolved spots in the field of view of each of the 10^8 lenses. This result is most encouraging, the principal problems ahead for this device now appear to be ones of quality control since it has been shown

that the lens and deflection system can be scaled down according to scaling laws. In the process of scaling down, the bits resolved per lens and the current density at the recording plane are constant while the bit density decreases as the square of the scale factor and the device size decreases by a factor between the square and the cube of the scale factor. (Obviously, this process cannot be continued

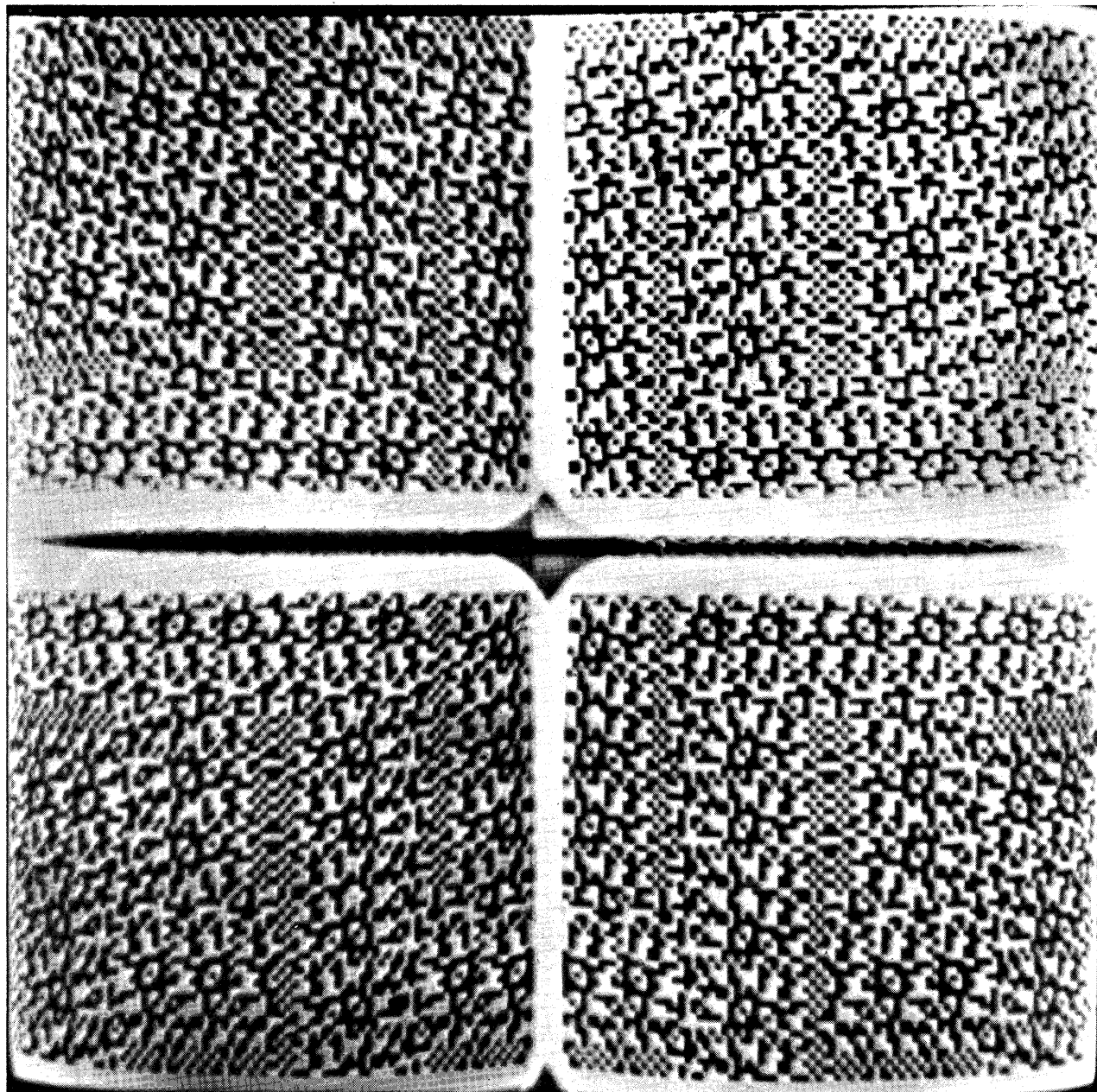


Figure 9. Image of simulated digital data from four neighboring lenslets of 32 by 32 matrix, illuminated by a single large-diameter beam. Data squares are 10 microns high while scan lines are less than one micron on two micron centers. Resolution is limited by the optical read-back system rather than the recording.



Figure 10. Portion of live image recording on photographic film. Scanning beam less than 1 micron on 2-micron centers.

indefinitely without increasing current density since one would eventually reach a limitation due to the statistical inadequacy of the electron beam, because while current density is maintained at the same level, ampere seconds per spot is reduced. We are well removed from that limitation in present considerations, however.) Thus each of the 1000 small lenses should ultimately be capable of recording the same number of bits per field of view as a large lens which certainly approaches 10^8 bits. The advantage

is that 1000 of these single tube memories are contained in one small device with a small number of input leads.

In summary, we have described a novel electron optical element which permits the entire memory plane to be in focus at one time. The current density at the recording plane is maintained and the memory plate size is reduced by the square of the scale factor. A packing density of 10^8 bits per square inch has already been demonstrated with 1 micron beam

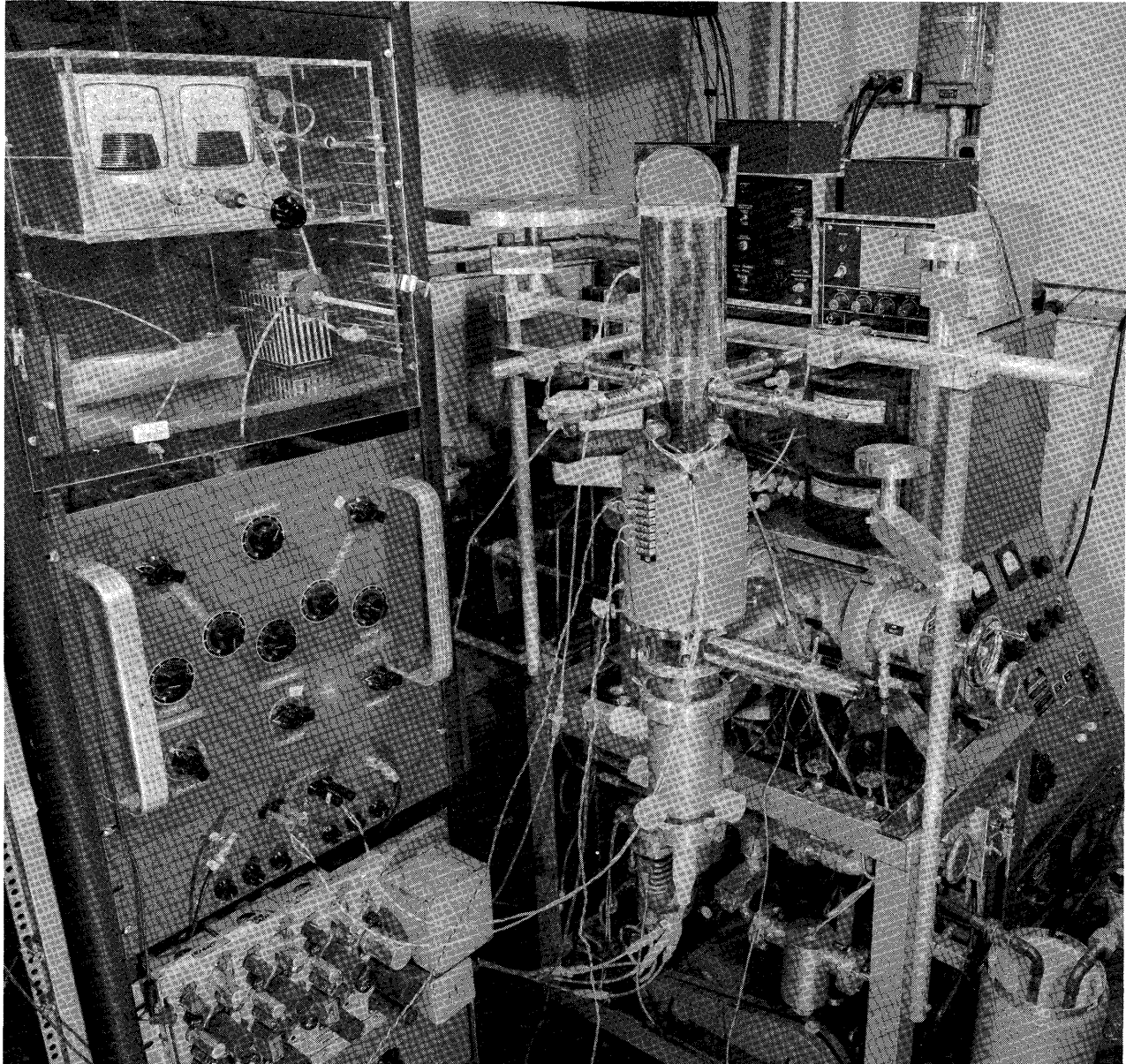


Figure 11. Improved test equipment for continued evaluation of 32 by 32 lens in submicron recording dimensions. Small electron microscope added at top for focusing electron image.

diameter. The next extension of performance will require an electron microscope to judge focus of the beam and to display the recording.

REFERENCES

1. R. K. Jurgen, "Technical Highlights of —58 Wescon," *Electronics*, vol. 31, p. 72 (Nov. 7, 1958).
2. S. P. Newberry, "Problems of Microspace Information Storage," *Proceedings Fourth Symposium on Electron Beam Technology* (R. Bakish, ed.), Alloyd Electronics Corporation, Cambridge, Mass., 1962, p. 81.
3. E. Ruska, "Experiments with Adjustable Magnetostatic Lenses," *Electron Physics*, National Bureau of Standards Circular 527, paper #44, p. 399 (1954).
4. S. P. Newberry and S. E. Summers, "The General Electric Shadow X-Ray Microscope," *Proceedings of Third International Conference on Electron Microscopy Held at London, July 1954*, Royal Microscopical Society London, 1956, paper #64, p. 305.
5. T. Hibi, "Pointed Filament and Its Applications," *ibid*, paper #151, p. 636.
6. M. M. Salpeter and L. Bachmann, "Autoradiography with the Electron Microscope," *J. Cell Biol.* vol. 22, p. 469 (1964).
7. C. E. Mees and T. H. James, *Theory of the Photographic Process*, 3d ed., Macmillan, New York, 1966, Chap. 2, p. 36.
8. J. F. Hamilton and L. E. Brady, *J. Appl. Phys.* vol. 30, p. 1893 (1959).

A SYSTEM OF RECORDING DIGITAL DATA ON PHOTOGRAPHIC FILM USING SUPERIMPOSED GRATING PATTERNS

R. L. Lamberts and G. C. Higgins

*Research Laboratories, Eastman Kodak Company
Rochester, New York*

The most common method for storing data in a binary form on photographic films has been to record each bit in terms of the presence or absence of a density in an area on the film. While in theory high-resolution materials have the inherent capacity for recording a tremendous number of bits within a given area, to a very great extent this has not been realizable because of technological difficulties. First of all, as the areas corresponding to the bits on the film are made smaller, they become more and more difficult to locate mechanically. And second, any film is bound to pick up a certain amount of dirt, and such particles can easily obscure individual bits and cause real havoc if high accuracy is required.

The method to be described represents a new system for recording such information, which largely circumvents both of these difficulties and hopefully makes data recording on film a much more practical sort of operation.

The basic principle of this system is to record a bit of information as a small grating pattern on the film. As shown in Fig. 1, when such a pattern is placed in a simple optical system and the slit is illuminated with monochromatic light, two first-order lines will be formed. We can now place photodetectors at these lines and determine whether the grating pattern is in the aperture or not. It is interesting and

important to note that the diffraction lines do not move when the grating pattern is moved.

So far we still do not have a very efficient recording method, but we can make it much more so by recording each bit in a given character as a pattern of a given spatial frequency and superposing the grating exposures on top of one another. Thus, for a character consisting of seven bits, we will now record a composite grating pattern consisting of up to seven spatial-frequency components. Figure 2 shows an enlarged composite grating consisting of seven components. The components are sinusoidal—or nearly so—in this case.

When such a pattern is placed into our optical system, a pair of first-order lines will be present in the focal plane for each component frequency of the composite pattern, as shown in Fig. 3. We can therefore place a photodetector at the position of each first-order component, on one of the sides at

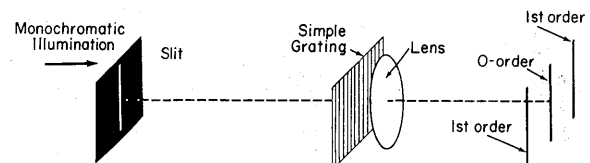


Figure 1. Optical system producing zero-order and first-order lines from a simple grating.

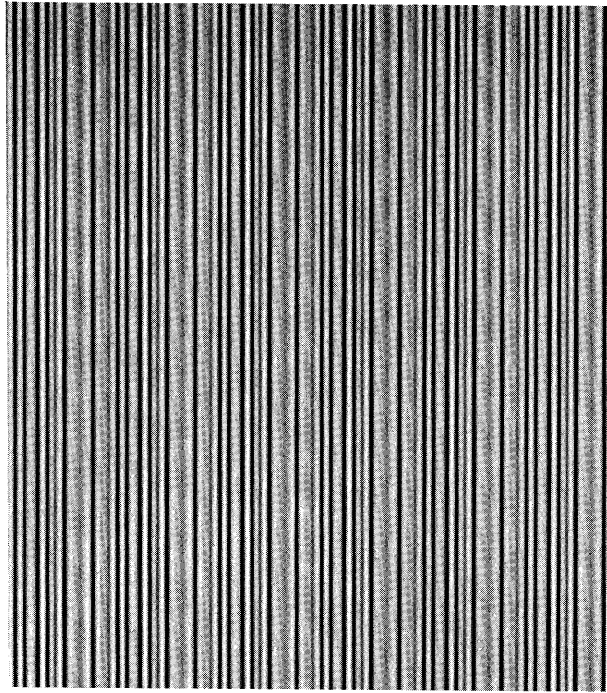


Figure 2. Enlarged composite grating pattern consisting of seven spatial frequency components.

least, and determine whether a spatial frequency has been recorded or not.

Figure 4 shows diagrammatically the position we would want for the first-order lines. On the basis of simple grating theory you can show that the distance between a given first-order component and the zero-order image will be directly proportional to the spatial frequency, which means that, for an even spacing of the lines, we want to have equal increments between the spatial-frequency values. Likewise, because there is a possibility that a second-order line will be formed, all the spatial frequencies should be contained within a single octave.

Figure 5 shows a series of actual photographs of diffraction patterns formed from various composite grating patterns. It is obvious that the lines are very distinct and that an unambiguous signal is formed.

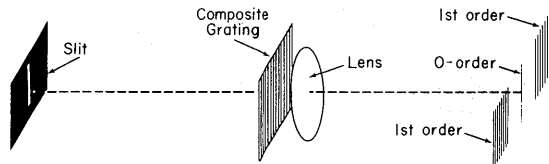


Figure 3. Optical system similar to that in Fig. 1, producing a number of first-order lines from a composite grating.

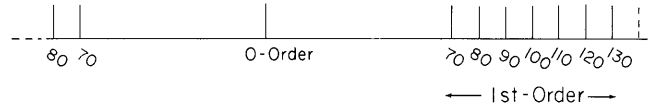


Figure 4. Diagram showing relative positions of diffraction lines.

The practical limit for the number of spatial frequencies within a given area appears to be about seven or eight since the line intensities drop off rather rapidly as the number is increased beyond this point. But with seven frequency components it is not uncommon to find that the stronger of the first orders have intensities of about 5% or even as high as 10% of the zero order.

It should be pointed out that if the information of seven bits spread uniformly over the area otherwise occupied by the seven density areas in the conventional system, this system should be less sensitive to dirt and scratches because the area for a given bit is at least seven times as large. For the same reason, the problem of mechanically locating an area is much less difficult. This system is similar to the basic approach to holography. In fact, it turns out that a composite grating is essentially a Fraunhofer-type hologram of a series of bright points.

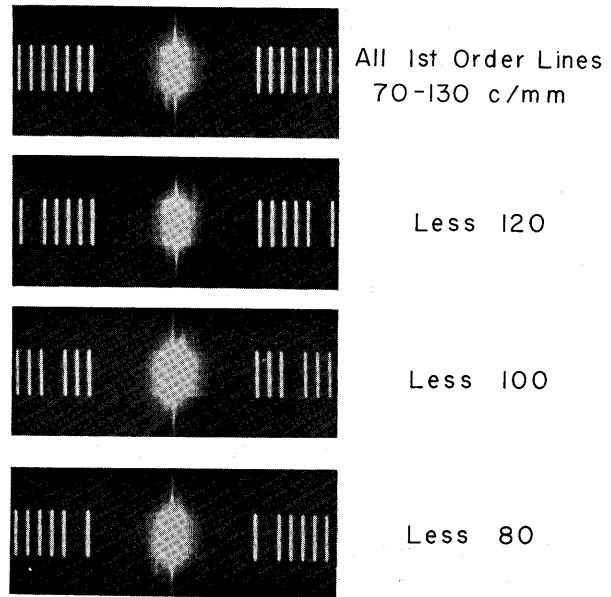


Figure 5. Photographs of multiple first-order lines produced from various composite grating patterns. The composite grating producing the upper illustration contained seven spatial frequencies ranging from 70 to 130 cycles/mm. The other illustrations were produced from grating patterns where one of the frequencies was omitted in each case.

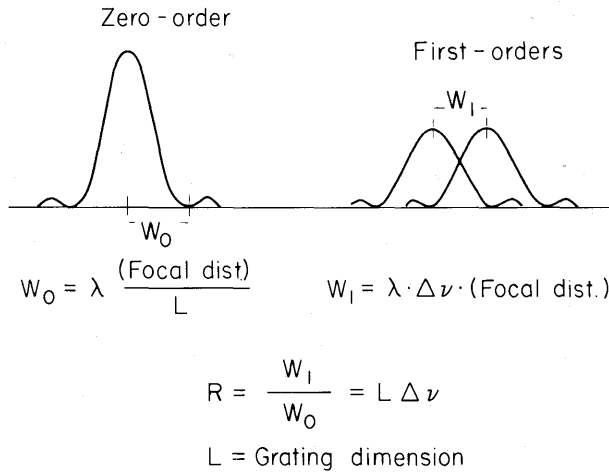


Figure 6. Zero-order and two adjacent first-order distributions illustrating the effect of redundancy.

While these patterns can be made very small, there is a limit to their size; this is illustrated in Fig. 6. Suppose that the composite grating is in the aperture plane of the optical system. There will then be a distribution of light at the zero order which is a diffraction pattern whose size, w_0 , will vary inversely with L , the size of the aperture, as shown by the equation at the left. The equation on the right, obtained from grating theory, gives the distance between the centers of two adjacent first-order lines. Since for well-formed gratings, the first-order components will have the same width distributions as for the zero order, we can define a redundancy factor R as the ratio of w_1 to w_0 . When R is equal to unity the two adjacent first-order lines will be just at the limit of resolution and w_1 will be equal to w_0 . This would occur, for example, if the pattern dimension were 1/10 mm and the spatial frequency increment were 10 cycles per mm. Thus by increasing either L or $\Delta \nu$, we are able to increase the resolution of the lines. Practical experience as well as calculations have shown that obstructions such as dirt particles can cause difficulties for small values of R , but as it

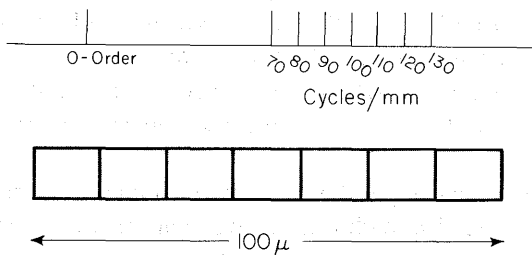


Figure 7. Comparison of storage for diffraction and conventional photographic systems.

is increased to five or more, the probability of false readings is greatly diminished.

To give just a little more insight as to how this system works, suppose that a set of seven spatial frequencies ranging from 70 to 130 cycles/mm was recorded as shown in Fig. 7. The basic interval between spatial frequencies would again be 10 cycles/mm. As has been pointed out, for a redundancy of unity the pattern would have to be 1/10 mm or 100 μ long. Suppose now that these same bits are recorded using clear and opaque areas in this same dimension of 100 μ , as shown in the lower part of Fig. 7. Then it would be necessary to have a resolution of at least 7 lines per 100 μ or 70 lines/mm, which is of the same general magnitude as is required by the diffraction system.

To increase redundancy with the *conventional* system, the clear and opaque areas would be made larger, which is essentially throwing away the resolution available in the system. In the *diffraction* system, the resolution is used when the pattern is enlarged. Many fine-grained films have a capability of resolving many times what has been required for most conventional film data-storage systems, but with the new system we have means for putting it to use.

One method of forming very good composite images is with the use of variable-area composite patterns of the type shown in Fig. 8. These can be photographed with a camera system using a cylindrical lens in front of the objective lens so that the image illuminance is made to vary as the composite function.

Both of the patterns shown in the figure are composites of the same seven spatial frequencies. The difference between the two is that the component

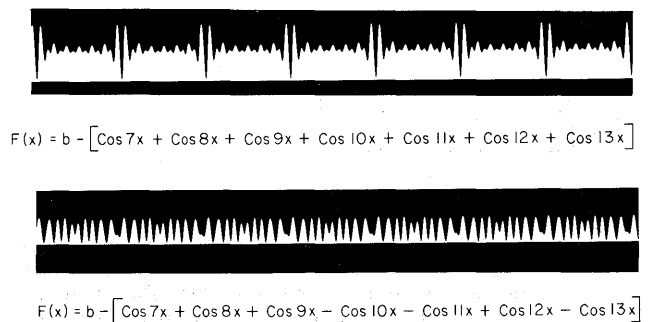


Figure 8. Composite variable-area patterns consisting of the same seven sinusoidal components but phased to give maximum and minimum variation in width.

spatial frequencies in the lower one have been phased to give a minimum overall amplitude while this was not done in the upper one. The width as a function of distance is given in each case by the equation below the patterns.

It can be seen that the first term in each of these is the constant b which represents a "bias" or average light level that is necessary to prevent the function from becoming negative. In the situation shown at the bottom of the figure, the bias level can be reduced to less than one-half of that shown at the top. To a first-order approximation, the intensity of the diffracted light that is monitored increases as the square of modulation of the component spatial frequency, and this would mean that it would increase as the square of the reciprocal of the bias level—that is, as $(1/b)^2$. Therefore, one should be able to increase the light approximately by a factor of four by properly phasing the patterns and reducing the bias level.

A method more adaptable for recording grating patterns is illustrated in Fig. 9. In this system, as the spot sweeps across the face of the cathode-ray tube, the intensity is modulated by any combination of the seven oscillators shown schematically in the figure. The temporal frequency of the oscillator, of course, will control the spatial frequency of the pattern as it appears on the cathode-ray tube. Each oscillator then controls one bit of information in a given char-

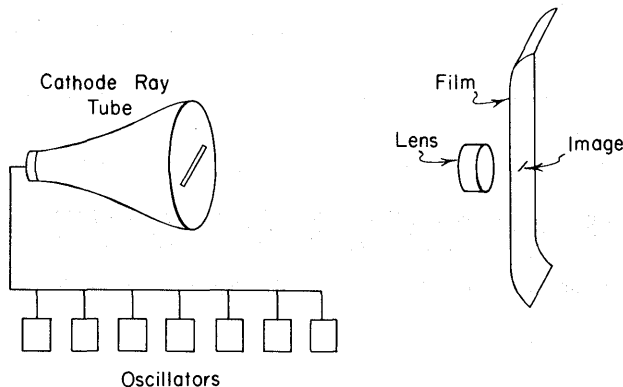


Figure 9. Sketch of cathode-ray system for recording composite grating patterns onto film. The oscillators serve to modulate the spot intensity as it sweeps across the face of the tube. Not shown in the figure is a cylindrical lens of comparatively high power approximately a focal length away from the tube face. Since this serves to increase the height of the image, it can be masked for producing an image of any desired height and of uniform intensity in the vertical direction.

acter and a composite pattern can be made on the tube face by summing up the outputs of the oscillators.

The patterns are then photographed onto the film as it moves in a film transport. The height of the patterns can be controlled by placing a cylindrical lens in front of the tube face to image the line into the aperture of the lens. By orienting this cylindrical lens so that its refraction is perpendicular to the direction of scan line, the whole lens will appear uniformly bright at the objective lens. The cylinder can then be masked to the desired dimension.

One distinct advantage of this type of read-in system is that, since the patterns are superimposed as electrical signals, the bias can be adjusted to any desired value. Experience has shown that the strength of the first-order diffraction patterns can be increased considerably by first obtaining proper phasing of the patterns to reduce the overall amplitude of the composite signal itself while not reducing the amplitudes of the individual components, and then reducing the bias level to as low a value as possible, as was shown with the variable area patterns. In fact, a certain amount can be gained by reducing the bias to the point where a considerable amount of clipping takes place at the low intensity portion of the image.

With this sort of apparatus, it has been possible to record at a rate of about 15,000 patterns per second. The dimensions of each of the patterns were 500 by 7μ . The long dimension was in the direction perpendicular to the lines. Up to seven spatial frequencies ranging from 70 to 130 cycles/mm with a 10-cycle/mm increment were recorded in a pattern area. Thus the redundancy factor was equal to 5—that is, 10 cycles/mm times 500μ .

With pattern dimensions of 500 by 7 microns and with the characters separated by 7μ , the packing density is about 6×10^5 bits/in².

The narrow dimension of the pattern is essentially limited by the resolution of the system, and excellent diffraction patterns can still be obtained from very narrow grating patterns. However, increasing this dimension does serve to increase the system redundancy.

A second type of recording technique is illustrated in Fig. 10. With this arrangement of prisms, individual grating patterns are illuminated from separately controlled sources, only one of which is

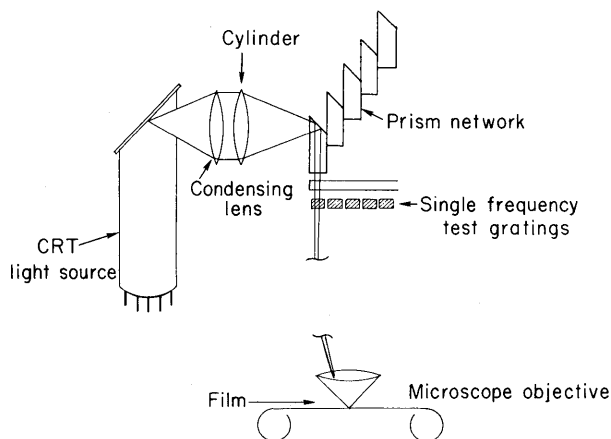


Figure 10. Prism array for optical addition of gratings. Only one light source is shown.

shown, and are imaged onto moving film with a single high-quality lens. Composite gratings can be produced through sequential addition of the individual gratings by gating the light sources in synchronism with film motion. However, memory and logic devices are required to maintain proper relations between coded inputs and photographic characters.

With emulsions having modulation transfer function characteristics similar to those of Recordak Micro-File AHU Film, Type 7459, the number of elemental gratings (binary ones) is practically limited to seven per photographic character. However, to allow a sufficient number of bits for timing and error correction in addition to those required for digital information, it is desirable to provide as many as ten bits per character. This is readily accomplished by means of a dual-track format in which each track contains up to five bits.

Of several possible light sources, the Sylvania Modulated Light Source SC4079P-11 has been found best suited to the present application. It consists essentially of a simple, high-current electron gun with electrostatic focusing but without deflection electrodes. The phosphor is coated on a metal face plate positioned at 45° to the tube axis. By collecting radiation from the face of the phosphor on which electrons impinge, a considerable increase in radiant output over a conventional CRT is obtained. Radiation patterns show that in a direction normal to the tube axis the intensity is about half that in a direction normal to the face plate. For this reason the tube axis was generally oriented at 45° to the position shown in Fig. 10.

In other respects the light source exhibits the same characteristics as conventional CRT's and a compromise must be made between photographic speed and the grating quality that can be obtained with available optical elements. Phosphor persistence is such that modulation rates in excess of 100 kc are reasonable. However, the variation of effective spot diameter with grid modulation places a more severe restriction on recording rates. At low and moderate levels of brightness the spot diameter is relatively small so that only the central zone of the objective lens is illuminated. Image quality is excellent but recording rates are low. As the brightness is increased for higher speeds, the spot diameter also increases, and the outer zones of the lens, which are prone to spherical aberration, are illuminated. The problem was minimized by using a 10-mm microscope objective operating at about $f/1.2$ and by restricting spatial frequencies to the 50–90 lines/mm octave. In this frequency range and at a magnification of 30:1, adequate exposure on Micro-File AHU Film, Type 7459, was obtained at character rates up to 60 kc.

A system for reading out small pattern areas such as these is shown schematically in Fig. 11. This system makes use of a high-pressure mercury arc. While the level of the diffracted light is rather low, it is sufficient to operate photomultiplier tubes. A cylindrical lens can be used beyond the film to reduce the first-order lines into small spots of light. As noted, an important feature of this type of reading out is that the position of the diffraction lines does not move when the grating pattern moves. This greatly simplifies tracking since with a 500 micron long grating for example, lateral movement of the record by as much as 25 microns or one mil can be tolerated without significantly affecting the signal strength. The motion simply effectively shortens the grating.

A much higher light intensity can be obtained by using a helium neon laser as a light source, as shown in Fig. 12. In this case the diffracted light intensity can be used to operate photodiodes without difficulty at frequencies approaching 100 kc.

While it will take considerably more work to make systems of this type that are operationally foolproof, experience has indicated that the principles of operation are sound and that by using this type of system, much of the high-storage potential of fine-grained photographic materials can eventually be realized.

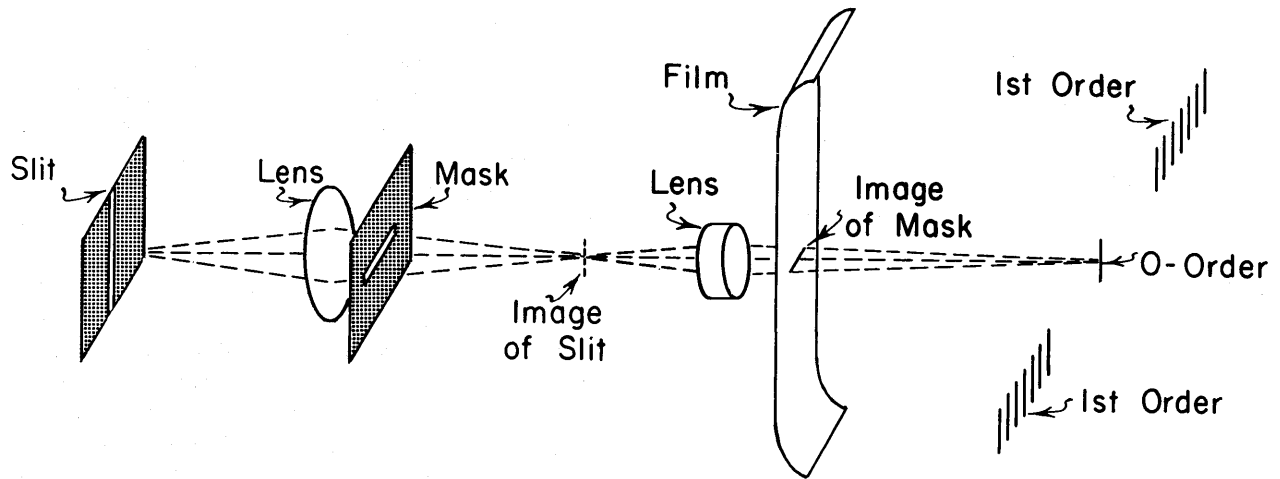


Figure 11. Readout optical system for illuminating a small area of the film. The nearly monochromatic green line of a high-pressure mercury lamp is imaged onto the slit at the left, which, in turn, is imaged by both lenses in succession. The mask is imaged by the second lens onto the film and serves to limit the illuminated area.

BIBLIOGRAPHY

Lamberts, R. L., and G. C. Higgins, "Digital Data Recording on Film By Using Superposed Grating Patterns. I. General Theory and Procedures," *Phot. Sci. Eng.*, vol. 10, no. 4, pp. 209-13.

—, "Digital Data Recording on Film By Using

Superposed Grating Patterns. II. Analysis of the System," *ibid.*, pp. 213-21 (1966).

Blackmer, L. L., A. P. VanKerkhove and R. Baldwin, "Digital Data Recording on Film By Using Superposed Grating Patterns. III. Recording and Retrieval Techniques," *ibid.*, no. 5 (1966).

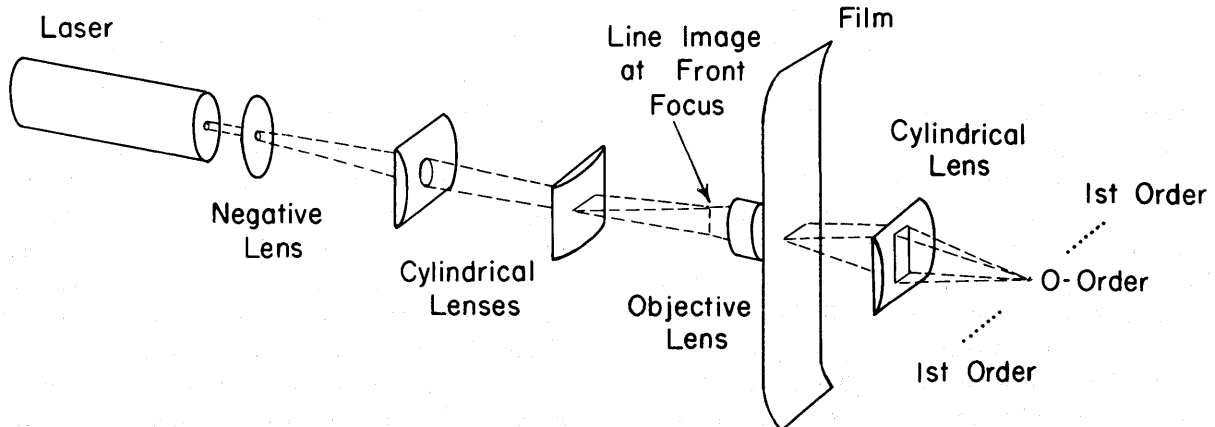


Figure 12. Read-out optical system with a helium-neon laser for illumination. A mask for controlling the width of the illuminated area can be placed at the second cylindrical lens. The height of this area can be controlled by the f -number of the objective lens and its focus.

A PHOTO-DIGITAL MASS STORAGE SYSTEM

J. D. Kuehler

IBM Corporation, Harrison, New York

and

H. R. Kerby

IBM Corporation, San Jose, California

INTRODUCTION

The photo-digital mass storage system does not impinge upon present day direct-access storage devices. Rather, it complements them and becomes another member in a hierarchy of files, each with its own specific capabilities and restrictions.

On-line ultra-high capacity is the major reason for the system's existence. Relatively slow access speeds presently make it less attractive in rapid response inquiry applications, and the nonerasable medium requires new consideration in applications involving highly volatile data such as is usually found in accounting applications. A number of applications are just now reaching a sufficiently defined systems maturity to warrant total automation, and are characterized by masses of fixed or low-volatility data required on an iterative, random basis.

In essence, the system is for applications where data requirements have been too voluminous to be justifiable on presently available direct access storage devices, but where activity ratios are so low as to negate the serial reading characteristics of magnetic tape.

Figure 1 shows the layout of the system. The sections which follow give a brief description of the

system, covering components and technology such as data storage, the system and data controllers, the writer (composed of the electron-beam recorder and the chip developer), and the reader.

DATA STORAGE

Data is stored on a 1.38 × 2.75-inch silver halide photographic film chip. Thirty-two chips, containing a total of approximately 150 million data bits, are stored in a small plastic cell. This cell is identical to the cell used in the IBM 1350 Photo Image Retrieval System. Thus, hardware similar to the IBM 1350 cell file, pneumatic transport (Fig. 2), and computer control processor systems is used to store and transport cells automatically to and from the data recording and reading stations.

The first cell file contains 2250 cells and houses the pneumatic blowers for cell transport. Additional files containing 4500 cells each may be added to increase total on-line capacity. A manual entry station on one or more of the files permits entry and removal of cells from the system. Any cell can be accessed and delivered through the pneumatic tube system to a reader in less than 3 seconds.

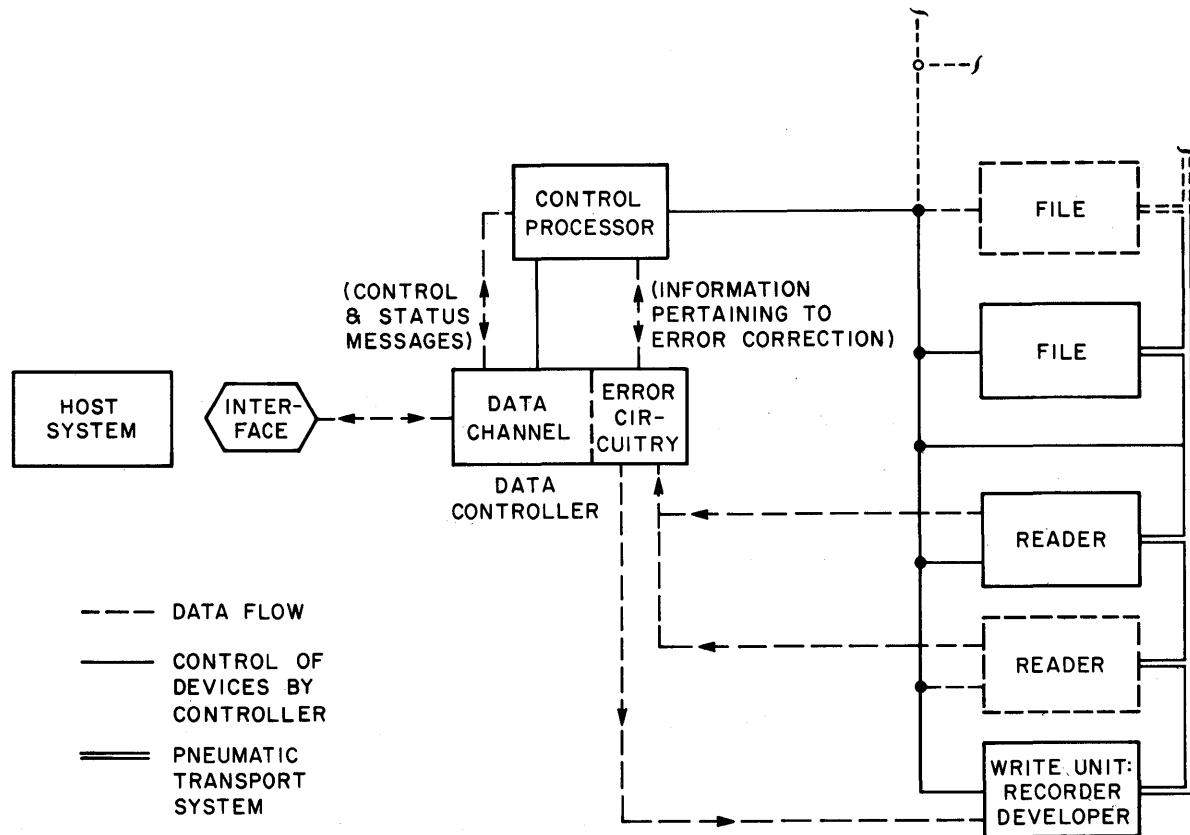


Figure 1. System block diagram.

SYSTEM CONTROL PROCESSOR

The system control processor serves as a digital process control computer with capability for error correction. Programs are stored and executed in the processor to directly control hardware functions in accordance with control and status messages received from the host computer via data controller. This includes input and output of data as requested by the host computer and special diagnostic and recovery programs used within the system. All hardware activities are controlled at the sense point and control solenoid level via special electronics in each hardware box.

DATA CONTROLLER

The data controller provides the complete interface to the host computer. It receives and transmits all control, response, and data messages and in turn interfaces with the control processor, recorder and reader for proper processing of the information. Core buffering and formatting circuitry is provided

to interface the host to recorder and host to reader. Special logic circuits encode error detection and correction bits with the data during recording and detect errors in the data during readback.

A multiple-burst independent character correction code is used to provide 6-character detection and 5-character correction over a 378-bit line. Eleven 6-bit characters of redundancy are used.

THE WRITER

Data is recorded on the silver film by using direct electron-beam exposure to produce a chip as shown in Fig. 3. Each of the 32 frames on a chip contains 492 tracks of data (246 track pairs). Each track contains 420 bits, serving a variety of functions, as shown in Fig. 3. Double-frequency recording is used; that is, a "one" is encoded as an "opaque-transparent" mark on the film, and a "zero" as a "transparent-opaque" mark.

Figure 4 shows the mechanical portion of the writer which is composed of the recorder and developer. It receives unexposed film, exposes it, and au-

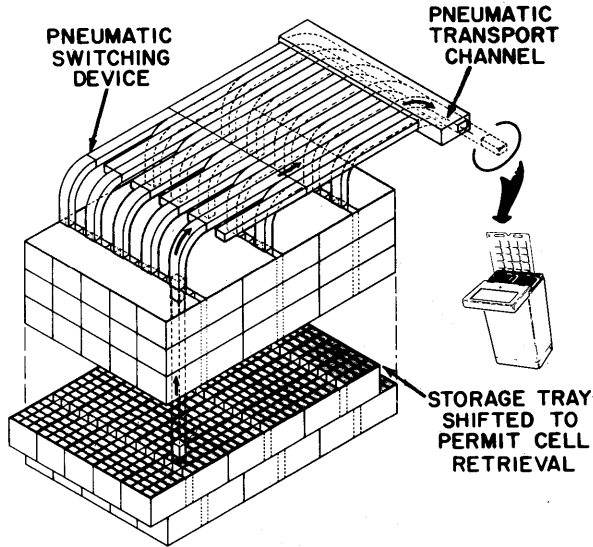


Figure 2. Cell storage and retrieval.

tomatically develops it in the on-line film processor. The completed chip is placed in a cell for delivery to the reader or file.

The writer can record and process 8×10^8 data bits per hour. A chip is ready for reading approximately 3 minutes after recording begins.

Recorder

Major components in the recorder section of the writer include: (1) the chip transport; (2) the chip format station; (3) the electron beam column and electronics; and (4) the vacuum pump system.

Chip Transport. Figure 4 depicts the transport as a rotary cantilevered chip-picking mechanism which removes an unexposed chip from a cell at the cell-handling station, rotates it, and places it in the chip format station. After recording, the transport places the chip in the developer, and after development returns it to a cell in the cell-handling station.

Chip Format Station. The chip format station serves to take the chip in and out of the vacuum system and position it before the electron beam (32 positions) so the data fields can be recorded. Figure 5 illustrates the principle of operation. The transport moves the chip through the slot in the top plates into a container attached to the lower plate. The lower assembly rotates approximately 120 degrees, and an intermediate vacuum removes most of the air from the container. Another 120-degree rotation (approximately) places the container underneath the slot in the upper plate leading to the high vacu-

um (low 10^{-4} torr range) recording chamber. The chip is pushed up into the chamber to 8 positions as the lower assembly rotates to 4 positions. After recording 32 frames, the chip returns to its container which rotates again, bringing the chip to the entry/exit slot. Thus, in normal operation, chip exit/entry, outgassing, and recording proceed simultaneously on different chips.

The lower plate is separated from the upper plate by approximately 0.001 inch, to minimize wear and seal problems. The vacuum system is designed to accommodate this differential pumping design.

*Electron-Beam Column and Electronics System.*¹ Figure 6 shows a schematic cross section of the

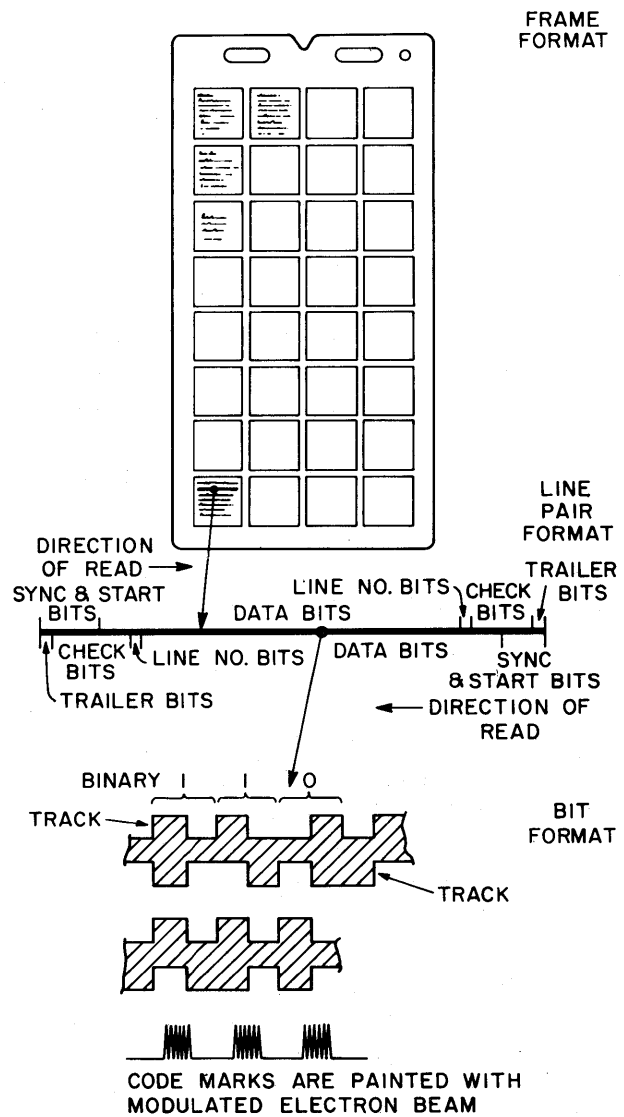


Figure 3. Chip and data format.

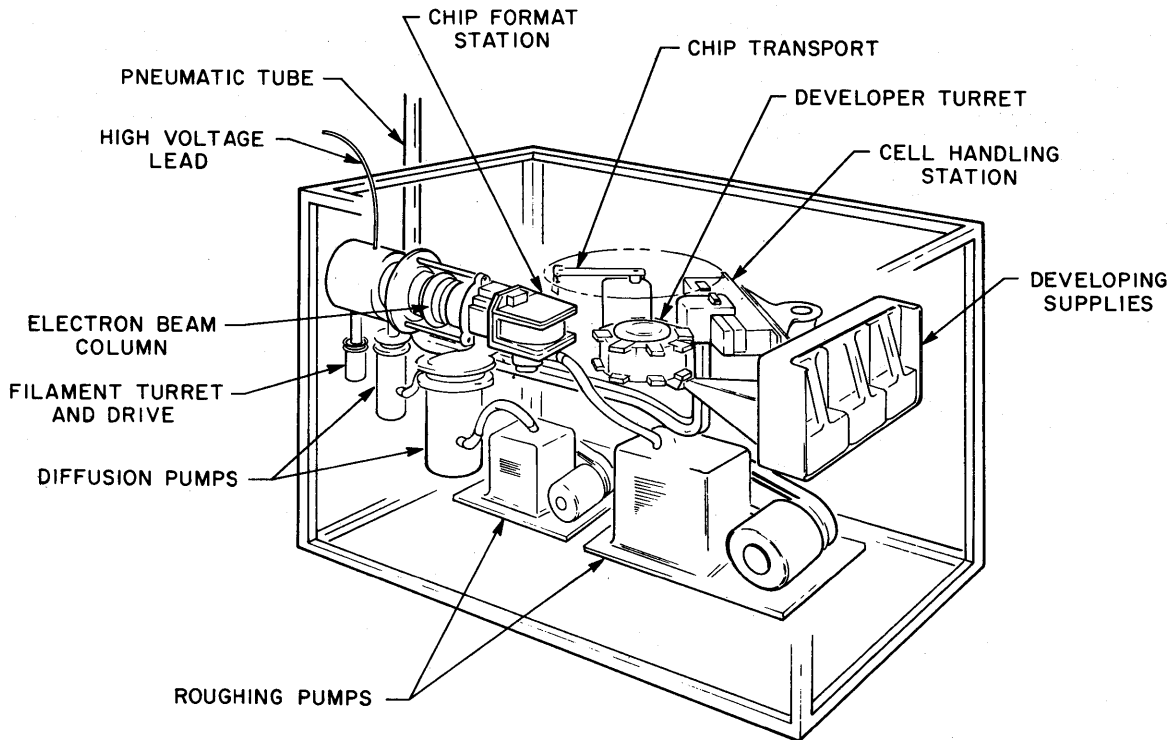


Figure 4. Photo-digital writer, containing the recorder and developer.

electron-beam column which delivers a controlled electron beam approximately 1.25 microns in diameter and 5×10^{-9} ampere. The beam repeatedly sweeps across a frame on the chip and "paints" the bits of information as shown in Fig. 3 at a rate of 5×10^5 bits per second.

The design of the electron-beam column is oriented toward unattended operation, minimum maintenance time, and ease of serviceability. One feature which illustrates this is the central precision tube (Fig. 6) that serves as a combination vacuum wall, alignment reference, electrical interconnector, and maintenance unit. This tube contains the beam-exposed parts, including the beam-sensing plates, apertures, pole pieces, and beam-blanking plates. Spring contacts on the internal parts and corresponding feedthroughs in the wall of the tube furnish the means for electrical connection to the outside. The whole prealigned assembly can easily be removed and quickly replaced by a new assembly.

Another feature which facilitates unattended operation and ease of maintenance is the turret of 16 tungsten hairpin filaments (Fig. 7) which quickly moves a new filament into place as needed. The turret allows a minimum of approximately 3 weeks

continuous operation before filament servicing is required.

To compensate for the differences between individual filaments and the effects of wire evaporation, the filament heating current is servo regulated. An a-c component in the heating current generates a signal proportional to the slope of the cathode-current versus heating-current curve. The very small signal does not produce significant beam current ripple or angular beam oscillation. By monitoring the slope, the filament servo corrects the heating current until the slope value is close to an adjustable reference value (Fig. 8) set for a minimum brightness of 5×10^4 amperes/centimeter² steradian at 12 kilovolts.

Four electronic servo systems are used to insure stable beam parameters:

1. A filament-operating-point control system provides stable long-life electron emission from each filament.
2. An alignment control system maintains the beam on axis to the column.
3. A spot-size control system periodically checks and minimizes the spot diameter in the plane of the film.

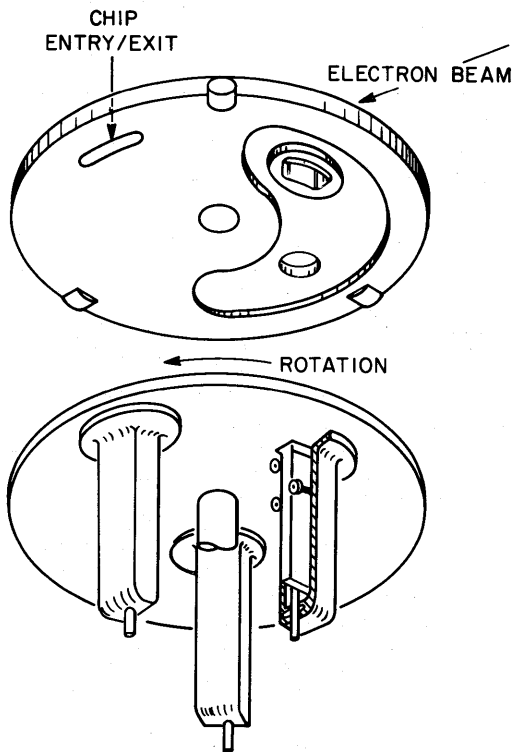


Figure 5. Format station.

4. A spot-current control system checks and adjusts the beam current arriving at the film, also on a periodic basis (between chips or less frequently).

An examination of the operation of the electron-beam system shows that, as the beam comes from the off-axial gun position, it is first bent by a deflection magnet into the tube axis to prevent filament light from arriving at the target. Filament emission is stabilized by the filament-operating-point control system described above. The beam is then partially collected by a symmetrical system of four electrically insulated sensing-plate quadrants that monitor the beam position. These sensing plates, beam heated to several hundred degrees centigrade to minimize contamination, provide the error signal for a servo system which automatically aligns the beam to the column axis via a magnetic alignment system.

Periodically a metal test target is placed in the recording plane for purposes of automatic spot-size measurement and adjustment. A solid state detector collects the beam transmitted through openings in the target. Using feedback control to an auxiliary focus control lens, the detector signal rise time, and

thus the spot diameter, is minimized to between 1 and 2 microns. A hold circuit maintains the focus control lens current constant between the periodic checks.

In conjunction with the spot-size adjustment procedure, the spot current arriving at the film is also adjusted to the desired value. This is accomplished by deflecting the beam into a Faraday cup for measurement and providing feedback control to lens one (nearest the filament). A hold circuit also maintains this lens current constant between periodic checks.

The nonmagnetic tube separates the pole pieces from the external lens. All apertures are heated to a temperature of about 300°C. This not only reduces contamination, but, as shown by Wilska,² also effectively and reproducibly renders most contaminants conductive. Electrical insulation of the apertures allows monitoring of the intercepted beam current.

An overall supervisory control system allows the system controller to monitor the operation of the

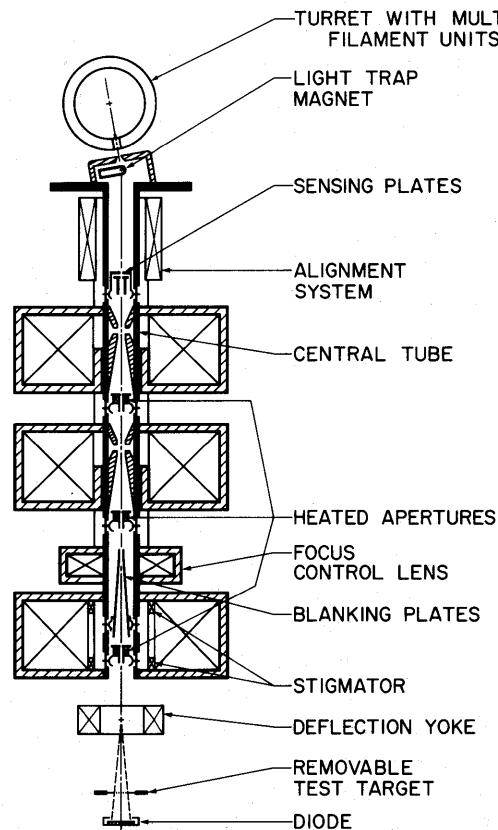


Figure 6. Cross section of the automatically stabilized electron-beam system. (Most supporting structures have been omitted.)

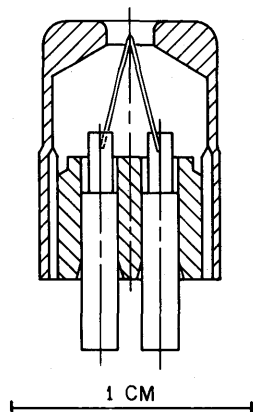


Figure 7. Filament unit used in the turret gun.

column and detect malfunction. A linear-sweep electromagnetic-deflection system is used in conjunction with an on/off high-frequency electrostatic-deflection system for painting each line of data. Line-to-line deflection is performed by using a precision digital-to-analog converter and electro-magnetic deflection.

Vacuum-Pump System. The basic components of the vacuum system include mechanical roughing pumps and diffusion pumps. One 5 cfm pump acts on the differential sealing slot of the chip format station. The pressure in this slot is 10 to 20 torr. A 4-inch diffusion pump, connected to the chip writing chamber, maintains a low 10^{-4} torr pressure. A 2-inch diffusion pump, connected to the filament turret, maintains a pressure in the low 10^{-5} torr range.

Pressure sensing is by means of both thermocouple (high pressure) and ionization (low pressure) gauges located in the filament and film-writing chambers.

Chip Developer

The developer accepts chips from the recorder at a maximum rate of 1 chip every 18.5 seconds and automatically processes and returns a dry chip of archival quality to the chip transport within a total cycle time of approximately 150 seconds.

Chips are processed individually in a small cavity (3cc). A set of 8 processor cells containing 1 cavity each comprises the rotary turret as shown in Fig. 9. The turret is indexed in 45-degree increments to sequentially present a processing cavity to a single fixed load-unload station and 7 individual and fixed processing stations. The processing cavities are sealed at each processing station by means of a ver-

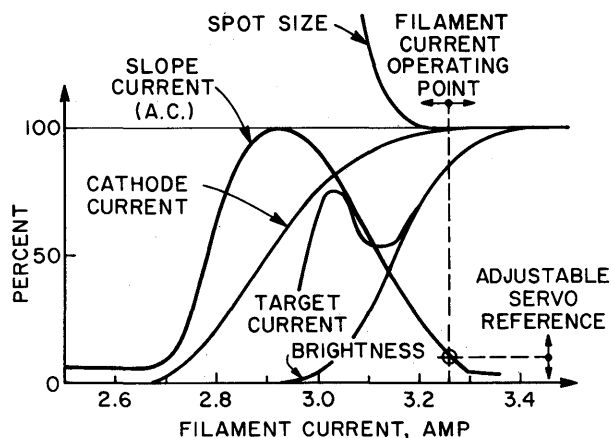


Figure 8. Various gun parameters as a function of filament-heating current. ("Slope current" is the a-c component of the cathode current generated by a constant a-c component in the filament-heating current. "Target current" is the current observed at the target under conditions of constant beam angle and demagnification. The curves apply for a new filament and shift to lower heating current with increasing filament age. By adjusting the reference of the filament servo system, the operating point and consequently the average life-time and brightness can be changed.)

tically reciprocating "cymbal" ring and deformable elastomer seals.

Chemicals are gravity-fed from 3 pairs of replaceable supply containers. After proper heating, air driven diaphragm pumps introduce a discrete volume (up to 5 cc) of chemicals into the processing cavities at the 2 chemical processing stations (develop/stop and fix/wash). Heated fresh water, introduced at 2 stations, insures complete washing; heated air, introduced at the final 3 stations, dries both chip and cavity and provides a heated cavity to the load-

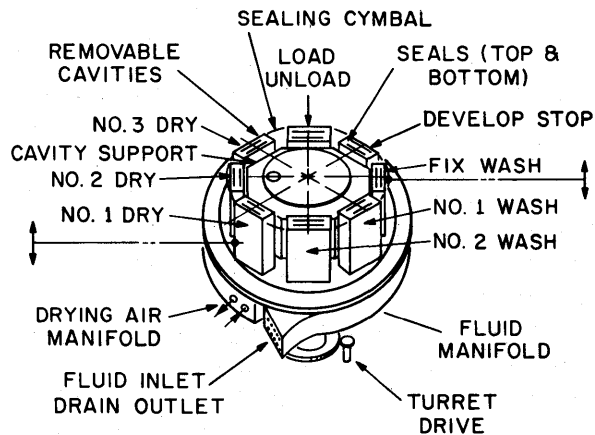


Figure 9. Rotary turret principle. The chip enters a developer cavity at the load-unload station and is rotated to the sequence of processor stations.

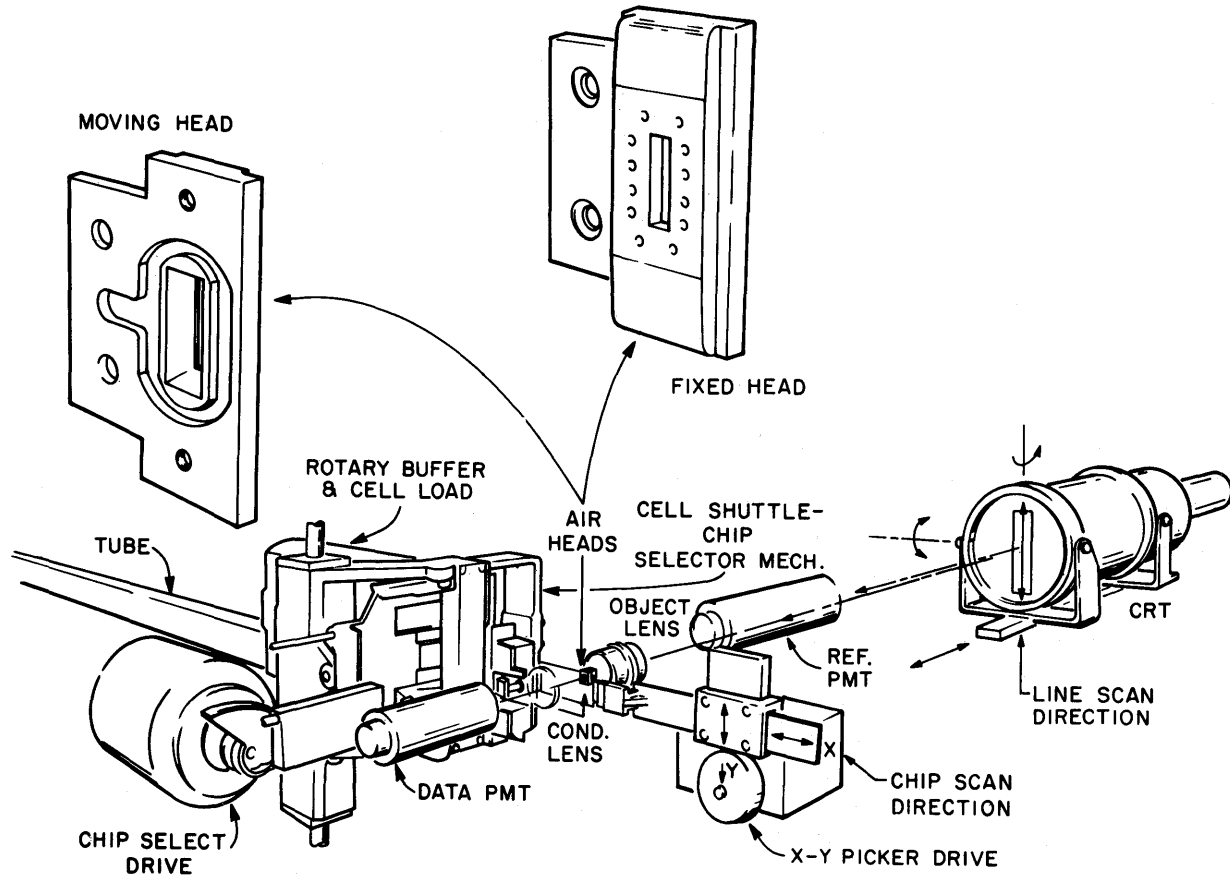


Figure 10. Photo-digital reader.

unload station. Individual temperature control is provided for the 2 chemical stations as well as for the wash water and drying air.

The developer does not reuse the chemicals. It is capable of operating under computer control for periods of up to 8 hours without operator intervention. Valving and level sensing means are provided in each of the 3 chemical supplies to permit automatic valving between active and reserve containers. The reserve supply can be replenished anytime during the last 4-hour period without interrupting developer operation.

READER

The reader (Fig. 10) uses a CRT (cathode ray tube) flying spot scanner to recover data from the film chip. Data is read at an instantaneous bit rate of approximately 2.5×10^6 bits per second. Accounting for the nondata bits read, the data bit rate is 2×10^6 bits per second within 1 chip. The multi-

chip sequential maximum read rate is approximately 1.1×10^6 data bits per second.

A cell arrives at the reader through the pneumatic tube and, via the cell and chip handling mechanisms, the chip to be read is addressed. The X-Y picker drive assembly picks the chip and rapidly moves it into the optics path between the two air heads. When the region of a frame at which reading is to begin is positioned within the "window" of the air heads, the chip stops and scanning can begin.

The CRT spot is imaged onto the chip by the objective lens, and the transmitted light is collected by the data PMT (photomultiplier tube). A line of data is read during each sweep of the CRT spot across the tube. Feedback from the data PMT provides CRT X-deflection to "capture" the spot on a data track. The scanner can loop on a pair of data tracks to provide reread and error-correction time. The scanner signals the X-Y drive to increment the chip for continuous reading of up to a full column (8 frames) of data.

A reference PMT is provided for a CRT intensity control loop and differential signal detection.

The 7-inch electromagnetic deflection CRT uses a P-16 short persistence phosphor to minimize phosphor decay time (scan speed approximately 34,000 ips). A spot diameter of less than 0.002 inch is maintained on the 5.7×1.5 inch scan area.

The $f/2$ objective lens has a magnification of 0.05. It images the flat face CRT to the film chip, with an image distortion of less than $\pm 0.5\%$ and minimum transmission of 70%.

Air heads provide a means of precisely positioning the emulsion plane of the chip at the image plane of the objective. The air bearings have a rectangular opening in the center to clear the optical path. Air is ported to the heads when a column selection is started, to provide a pneumatic guide as the chip enters the gap between the heads. When the chip reaches column 0, the moving head is pneumatically loaded to form a dual-pressurized air bearing, thereby precisely positioning the emulsion surface of the chip relative to the fixed air bearing. Positioning of the chip between the two air heads allows the chip to move in the X and Y direction in a plane perpendicular to the optical axis without scratching the chip.

The photomultipliers are $1\frac{1}{2}$ -inch S-25 tubes, with 10 stages of gain. Quantum efficiency of S-25 to P-16 phosphor is about 20%, and the tubes operate at a current gain of approximately 0.5×10^6 .

A closed-loop stepping motor system actuates the X-Y picker drive assembly. The key function of this system is that of rapidly advancing the film as needed during the reading process.

Special circuits developed for the reader provide the functions of spot deflection, servo control, line incrementing, spot size, spot brightness, and CRT-

PMT biasing. In addition, special detection and clocking circuitry accepts the phase modulated signal from the photomultiplier tube and converts it to clocked binary data. A variable-frequency clock, phased to the PMT signal, maintains proper operation even if data is lost for several bits or if the average data frequency varies.

A synchronous rectifier, integrator, and 3-level synchronous sampling technique produces a binary erasure channel output. In all, some 30 special solid-logic-technology card types are used.

SUMMARY

The photo-digital storage system, permitting the storage of great quantities of digital data, employs a method of data recording and handling which is new to the computer world. Basically, the storage unit is a silver halide film chip which holds about 5 million data bits. An electron beam records the data on the chips.

After the chip is recorded, it moves from the vacuum chamber to a developing station, and then to a storage cell. A pneumatic system transports the cell to a cell file, where it is held for retrieval and read-out.

The entire photo-digital operation is governed by a module controller. This controller directs the processing steps of the system in a manner similar to the automatic control of an industrial process.

REFERENCES

1. K. H. Loeffler, Sixth International Congress on Electron Microscopy, 1966.
2. A. T. Wilska, Fifth International Congress on Electron Microscopy, 1962, 1.D-6.

HYBRID COMPUTERS IN THE ANALYSIS OF FEEDBACK CONTROL SYSTEMS *

C. K. Sanathanan, J. C. Carter, L. T. Bryant
and L. W. Amiot

*Argonne National Laboratory
Argonne, Illinois*

INTRODUCTION

In the pursuit of accuracy, speed and economy in the simulation of mathematical models of physical systems, analysts are continuously searching for more sophisticated mathematical techniques and computer systems. When the complexity of the mathematical models increases, their simulations on pure analog or pure digital computers are frequently cumbersome. The main reason for this is that some of the operations in mathematical model are inherently best handled either on the digital or the analog computer.

A hybrid computer represents an effort to combine in one computer system some of the best characteristics of the analog and digital machines.^{1, 2} Generally, the intent in creating a hybrid is to combine the speed and efficiency of the analog in handling ordinary differential equations with the proficiency of the digital computer in performing logical and arithmetical operations.

The analysis of many physical systems involves the solution of partial differential equations. The analog computer cannot be employed to simulate these equations directly since it is limited to just one independent variable. Consequently, partial differential equations are simulated by discretizing all but one

of the independent variables. Since the number of the resulting ordinary differential equations increases as the product of the number of node points into which each of the independent variables is discretized, and since the analog is basically a parallel device, the simulation of models with several partial differential equations may require a prohibitive amount of analog equipment. In cases where sets of differential equations are similar, multiplexing of (time sharing) a block of analog circuitry for just one set of equations with the aid of the memory and logic capabilities of a digital computer can enhance the effective capacity of a given amount of computer equipment.³

In systems with internal feedbacks, however, multiplexing of computer equipment precludes the closing of the loop. This is explained as follows.

Consider the system represented by Fig. 1. Suppose the input (or forcing function) x^* is given. Let

* The input x and output y may consist of several components.

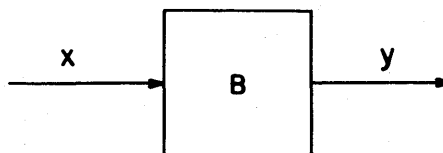


Figure 1. An open loop system.

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

B represent a system of equations which may be solved by multiplexing computer components. The output y of B may be obtained as a result of repeated application of the relevant part of the input x to the part of B which is actually simulated on a computer. An excellent example illustrating this approach may be found in Ref. 3. An important point to note here is that x is known (given) a priori; and is not affected by y .

In a system which has feedback, as shown in Fig. 2, x depends on y .

A simultaneous computation of all of y corresponding to x is necessary to close the loop. Multiplexing, however, results in a sequential solution of the sets of equations of B . Consequently, multiplexing in the simulation prevents the closing of the loop.

An iterative method is found to be effective in obtaining the closed loop system response.

The iterative process is illustrated in Fig. 3. It is initiated by using an arbitrary function y^0 . (In many situations y^0 may be taken as zero.) The convergence of the sequence, $y^0, y^1, \dots, y^i, y^{i+1}, \dots, y^n$ to the closed loop response, y , of the system in Fig. 2 depends upon characteristics of the equations in B , and in particular the boundedness of their variables. It is advisable to prove the convergence of the sequence before attempting the simulation.

The principle of contraction mapping is found to provide a basis for a proof of convergence of the iterative process. This gives a mathematical rationale for the extension of the technique of multiplexing to complex feedback systems.

The system considered in this paper for the purpose of illustrating the present approach to hybrid simulation is that represented by a fast neutron reactor core composed of ceramic fuel materials such as UO_2 . The mathematical model consists of differential equations describing the space and time-dependent neutron flux density, and the core material temperatures; and algebraic equations which give the feedback reactivity as a function of the transient temperature distributions.

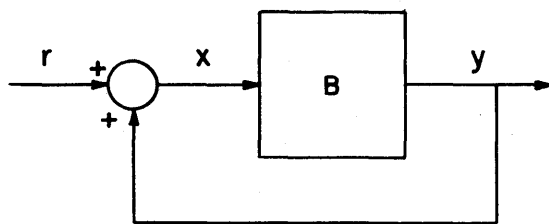


Figure 2. A system with feedback.

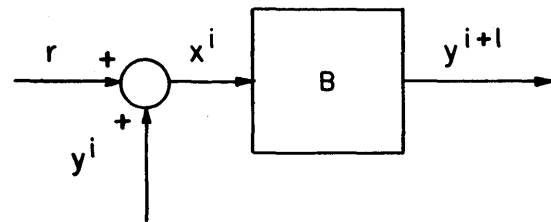


Figure 3. Iteration for the closed loop response.

The space dependence of the neutron flux is taken as invariant during transience. This assumption is realistic in reactors of moderate size. Consequently, the present treatment of the neutronics is macroscopic in nature.

The thermal properties of the ceramic materials change considerably with temperature,⁴⁻⁶ and the thermal gradients encountered are large. Due to these facts and since it is necessary to determine when and how phase changes (fuel melting, coolant boiling) occur in the core materials, no simple macroscopic approach to the determination of the core material temperatures is found to be realistic. It is therefore necessary to treat heat flow equations on a microscopic basis.

The detailed temperature profiles (radial and axial) in the various representative core regions are used to compute the feedback of reactivity.

Considering the nonlinearities of the differential equations of the system, they are simulated efficiently on analog components, multiplexing the circuitry wherever possible. Digital components are used for memory, for logic and for the arithmetic operations involved in the computation of the feedback reactivity.

The present method of hybrid simulation is proving to be very effective in the analysis and design studies of nuclear reactors. Since the techniques used are quite general, the method has wide application in complex feedback systems.

MATHEMATICAL MODEL

The equations which comprise the model of the reactor core describe the transient neutron flux density and the concomitant flow of the fission heat in the core materials. The temperature changes in the core affect the nuclear properties of the materials. The variations in the nuclear properties are reflected in the parameters of the equations for the transient neutron flux. Thus the model constitutes a feedback control system.

The equations of the model are stated below and discussed in Appendix 2.

Neutron Kinetics Equations

The neutron flux F is expressed as

$$F = \psi(\vec{r}) n(t) \quad (1)$$

$\psi(r)$, the space-dependent part* of F , is assumed to be known. $n(t)$ is given by Eq. (2).

$$\frac{dn(t)}{dt} = \frac{1}{l} (\rho - \beta) n(t) + \sum_{i=1}^6 \lambda_i c_i$$

$$\frac{dc_i}{dt} = \frac{\beta_i}{l} n(t) - \lambda_i c_i \quad (2)$$

$$i = 1 - 6$$

where l = the neutron lifetime,

β_i = the i th delayed neutron fraction,

c_i = the i th precursor concentration,

λ_i = the decay constant of c_i

$$\beta = \sum_{i=1}^6 \beta_i, \text{ and}$$

ρ = the excess reactivity.

$\rho(t)$ has two parts: the externally added reactivity, $\rho_{\text{ext}}(t)$ (the forcing function of the system); and $\rho_{\text{fb}}(t)$ the feedback reactivity.

$$\rho(t) = \rho_{\text{ext}}(t) + \rho_{\text{fb}}(t) \quad (3)$$

The feedback is that due to the temperature changes in the various materials of the core. The parameters l , β , and the λ 's are normally constant and so considered herein.

The different components of the feedback reactivity of any representative core region k are computed typically as:

$$\rho_{\text{fb}}^k = a \ln \left(1 + \frac{\Delta T_{\text{fuel}}^k}{T_0} \right) + b \Delta T_{\text{fuel}}^k + c \Delta T_{\text{coolant}}^k \quad (4)$$

where a is the Doppler coefficient and b and c are the coefficients of reactivity due to thermal expansion. ΔT_{fuel}^k and $\Delta T_{\text{coolant}}^k$ are the changes in the fuel and coolant average temperatures of the region k . T_0 is a reference temperature. The overall feedback is given by

$$\rho_{\text{fb}} = \sum w^k \rho_{\text{fb}}^k \quad (5)$$

* Note that \vec{r} refers to the entire core.

where w^k 's are the appropriate weights for the regions. w^k 's are known a priori.

The time-dependent part $n(t)$ of the neutron flux is obtained by solving Eq. (2). This, when multiplied by the space dependent part $\psi(r)$, gives the neutron flux in the reactor. This flux is multiplied by the fission cross section distribution in the core, $\Sigma_f(r)$, and the energy release per fission, E , to obtain the internal heat generation rate.

$$q(r,t) = [\psi(r) n(t)] [\Sigma_f(r)] E \quad (6)$$

Transient Heat Flow Equations

The reactor core is an assembly of fuel cells in the form of a right cylinder with the coolant flowing co-axially.

A typical fuel cell consists of a cylindrical rod of fissionable material with concentric cylinders of the bonding material, clad and coolant (Fig. 4).

Axial conduction of heat is neglected. Therefore, the heat flow equations are coupled in the z -direction, i.e., along the direction of flow, only through the variation of the coolant temperature along the z -axis. This temperature variation is dictated by the energy transport due to the coolant flow. An important consequence of this is that the equation for the radial heat flow is the same (in form) at any axial location.

The z -axis is divided into "small" increments Δz (Fig. 4). The radial temperature distribution in the fuel, bond and clad is considered to be invariant in the distance Δz . The coolant temperature, however, is assumed to vary linearly along Δz .

The radius is discretized as shown in Fig. 5. The equation of radial heat flow at any axial segment j is then †

$$c_p \frac{d}{dt} \left[\left(\frac{1}{8} - \frac{\Delta r}{24r_i} \right) T_{i-1}^j + \frac{3}{4} T_i^j + \left(\frac{1}{8} + \frac{\Delta r}{24r_i} \right) T_{i+1}^j \right]$$

$$= K_{i,i+1} \left(1 + \frac{1}{2} \frac{\Delta r}{r_i} \right) \frac{T_{i+1}^j - T_i^j}{(\Delta r)^2}$$

$$- K_{i-1,i} \left(1 - \frac{1}{2} \frac{\Delta r}{r_i} \right) \frac{T_i^j - T_{i-1}^j}{(\Delta r)^2} + \bar{F}_i^j n(t) \quad (7)$$

† Note that r in the present context refers to the radial coordinate of the fuel cell. See Ref. 7 for a detailed derivation of Eq. (7).

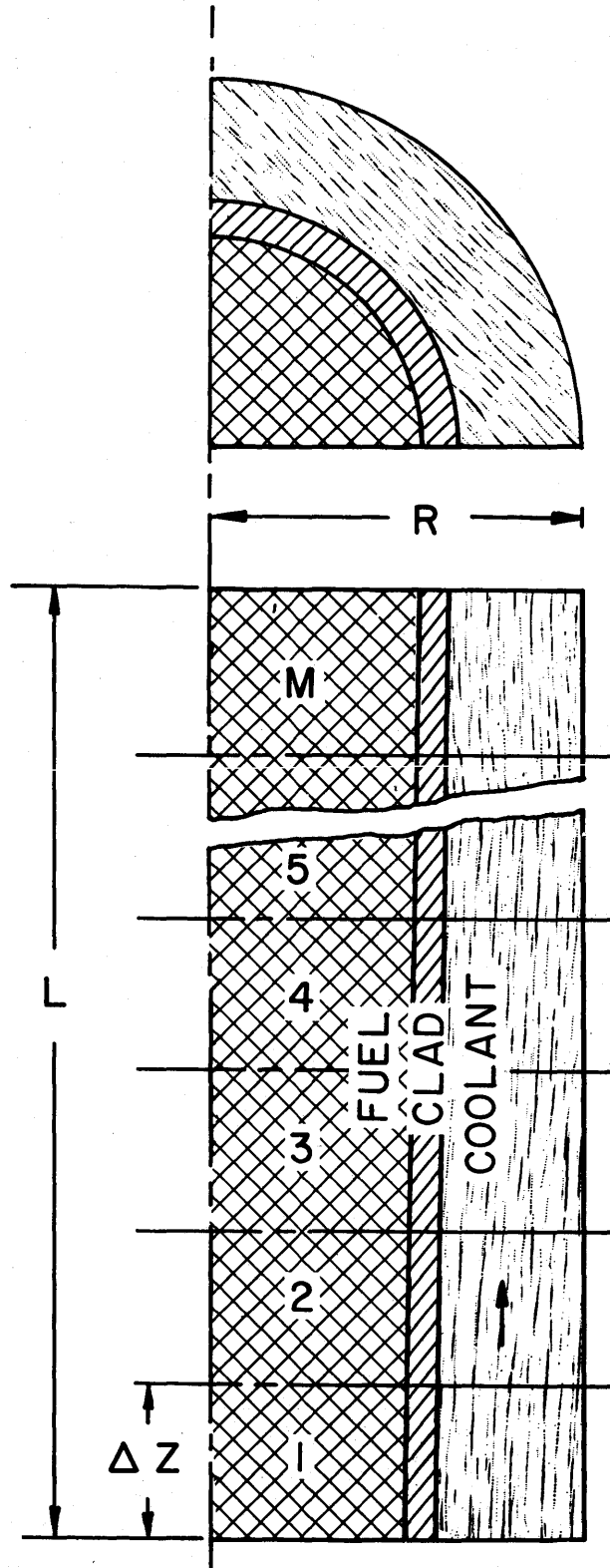


Figure 4. Model of a typical fuel cell.

The conductivity at the boundary between the radial sections i and $i+1$ denoted by $K_{i,i+1}$ is evaluated at the boundary temperature $T_{i,i+1}^j$. $\bar{F}_i^j N(t)$ represents the heat generation rate in the radial segment at r_i and it varies along the z -axis.

The boundary conditions are:

$$T_0^j = T_1^j \tag{8}$$

and

$$-K_N(T_N^j) \frac{T_N^j - T_{N-1}^j}{\Delta r/2} = h(T_N^j - \bar{T}_c^j) \equiv Q^j \tag{9}$$

\bar{T}_c^j is the average coolant temperature at the j th axial segment and h the heat transfer coefficient.

The conservation of energy in the coolant results in the following equation.

$$\frac{dT_c^j}{dt} + \frac{2}{\rho \Delta Z} G(t) [T_c^j - T_c^{j-1}] = \frac{\Lambda}{\rho c} Q^j \tag{10}$$

where $j = 1 \dots M$,

Λ = the ratio between the circumference of the clad and the cross section area of the coolant flow,

$G(t)$ = the mass flow rate of coolant per unit area, and

T_c^{j-1} = the inlet coolant temperature to the axial segment j . Its exit temperature T_c^j is computed from

$$T_c^j = 2 \bar{T}_c^j - T_c^{j-1} \tag{11}$$

The inlet coolant temperature of the core, T_c^0 , is a known function of time.

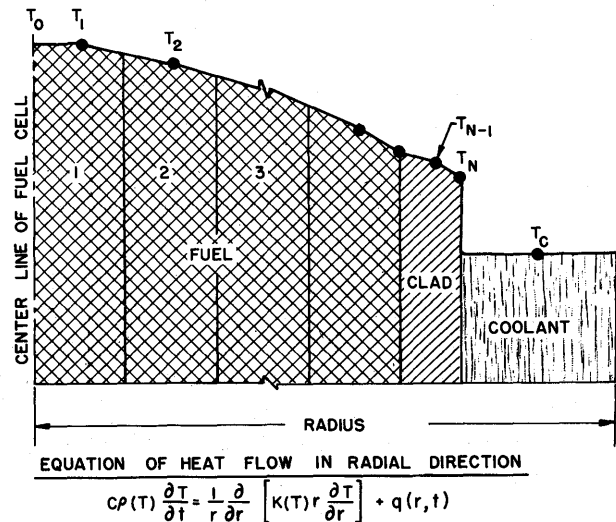


Figure 5. Radial nodes.

The equations developed thus far define the mathematical model. With this model, the various phenomena are described quantitatively. To summarize, Eq. (2) gives the time-dependent part $n(t)$ of the neutron flux, which, when multiplied by $\psi(\vec{r})$, gives the space time dependent neutron flux. Equation (6) gives the concomitant fission heat release in the nuclear fuel. From the knowledge of the internal heat generation rate in each fuel cell, the corresponding two-dimensional transient temperature profiles are obtained by solving Eqs. (7) to (11). Note that $N \times M$ number of equations of the form (7), M number of equations of the form (10), and the boundary conditions have to be solved simultaneously to obtain the detailed temperature profile in each cell.* Finally, Eqs. (4) and (5) give the transient feedback of reactivity.

The resulting closed loop system may be represented by the block diagram in Fig. 6. The simulation of this system may be made on a pure analog or a pure digital computer. However, it is found that the simulation on a hybrid computer has some outstanding advantages in terms of economy in computer equipment and programming effort.

HYBRID SIMULATION

From the mathematical model just described, it is apparent that a major portion of computing is in the solution of the differential equations of heat flow (Eqs. (7) to (11), $(N + 1) \times M$ in number) to obtain the two-dimensional temperature profiles in the fuel cells. Although the analog computer is efficient in handling these ordinary differential equations, the computing equipment required is enormous. From the nature of these equations, however, it is clear that the equations for the radial temperature profile are identical in form at any axial increment. This immediately suggests multiplexing of analog circuitry.

In the present simulation, therefore, the heat flow equations are programmed only for one axial segment in one fuel cell. This requires only the simulation of N equations of heat conduction ($i = 1 \dots N$ in Eq. (7) for any j) and one equation for the temperature rise in the coolant in the length Δz (Eq. (10) for any j). Figure 7 gives the detailed analog circuitry. Since axial conduction of heat is neglected,

* A value of 10 each for M and N is reasonable in many practical cases. Simulation of 5 representative fuel cells is considered adequate to describe the entire core.

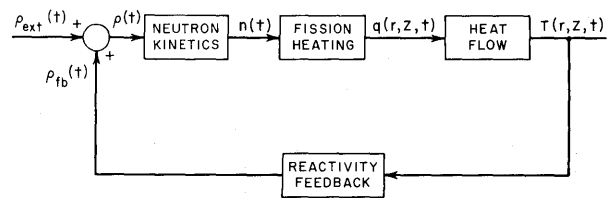


Figure 6. Block diagram of the mathematical model.

the successive axial increments are thermally coupled only through the flowing coolant. The exit coolant temperature of the axial segment j is the inlet temperature of the segment $j + 1$. Therefore, the thermal coupling between axial segments is only through Eq. (11).

The two-dimensional temperature profile in any fuel cell is determined as follows. The inlet coolant temperature, T_c^0 , and the flow rate G are known functions of time. The radial temperature profile of the first axial increment, and the coolant temperature T_c^1 at the exit of this increment are computed on the analog.

T_c^1 is obtained as an analog voltage. This voltage is sampled at a rapid rate and stored in the memory of the digital computer unit.

T_c^1 is made available from the digital memory, as the inlet temperature of the second increment and the same analog circuitry is used again to compute the radial temperature profile of the second axial segment.

The above process is continued for all axial increments.

Thus, the incorporation of a memory unit to the analog computer has made possible a repetitive use of just one set of the radial heat flow equations to cover the entire z -axis.

The other fuel cells are treated sequentially in the same manner.

If axial conduction of heat is not negligible, the determination of the two dimensional temperature profile is accomplished by repeating the above computational procedure several times. Initially the profiles are determined for no axial heat flow. From the axial profile of temperature thus determined the axial heat flow between segments is computed. Using this information, new radial and axial profiles are determined. The process is repeated until convergence is achieved.

The simulation is completed by programming Eq. (2) for the time dependent part of the neutron flux, $n(t)$, and Eqs. (4) and (5) for the temperature-dependent feedback reactivity.

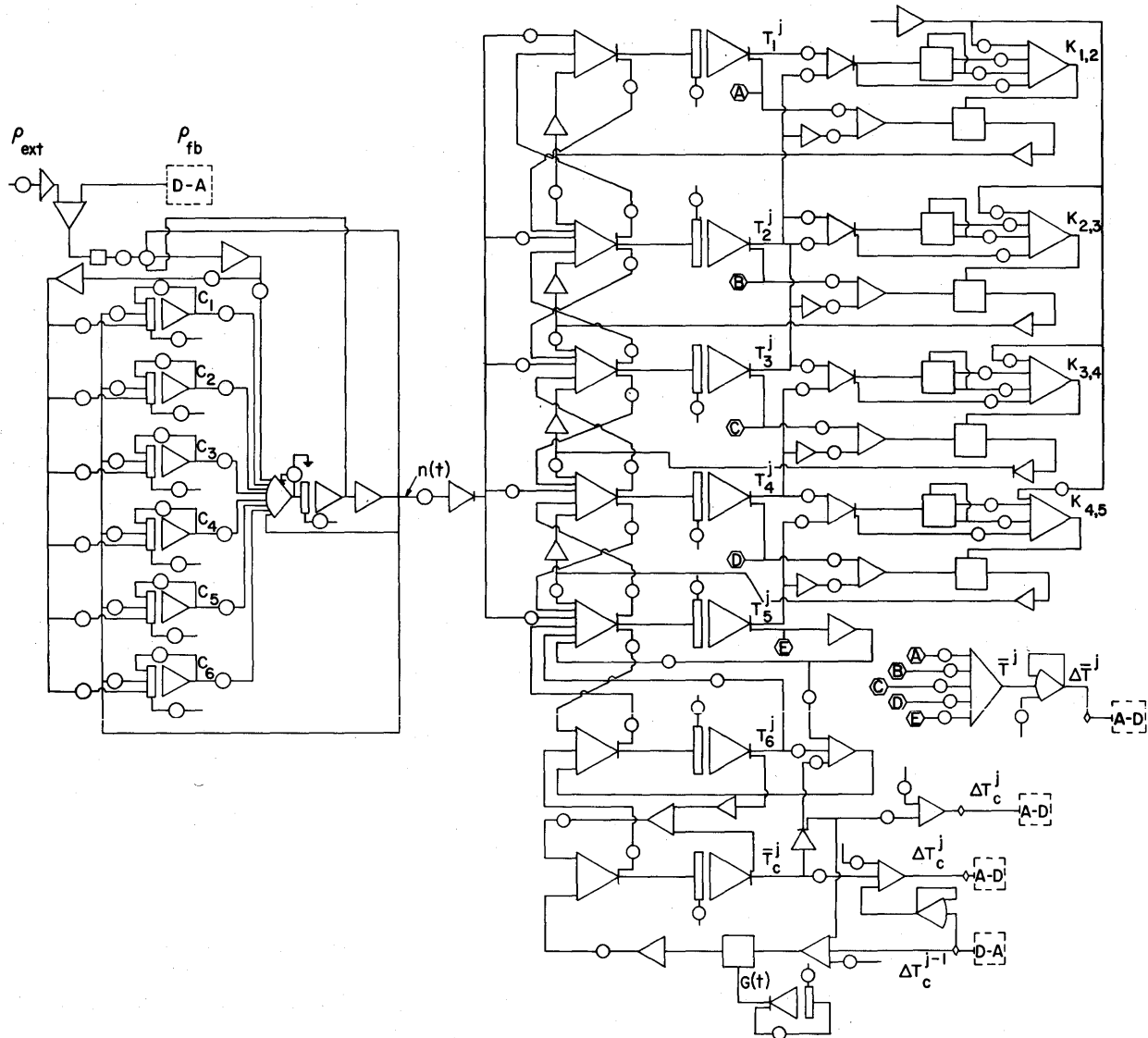


Figure 7. Multiplexed analog circuitry.

The analog computer is conveniently used to simulate the neutron kinetics equations.

Since the equations for the feedback reactivity are algebraic in nature, and since the various temperatures may be conveniently sampled,* and stored in the digital memory, the digital computer is the logical choice to handle the feedback equations.

The response of the closed loop system (shown in Fig. 6) is obtained by means of the iterative process described in the Introduction and illustrated in Fig. 3.

The iteration is initiated by using an arbitrary

* A discussion of the sampling procedure is given below in paragraph beginning "Sampling rates . . ."

feedback reactivity (as a function of time). The corresponding transient neutron flux and the temperature profiles are determined using the methods already described. The digital component now calculates a new feedback reactivity concomitant with these temperature profiles. This feedback reactivity is used for the next iteration. The iterative process is continued until the required convergence of the system variables is achieved. The schematic of the hybrid computer is given in Fig. 8. The computer flow chart is presented in Fig. 9.

The neutron kinetics in Eq. (2) are very sensitive to the net amount of reactivity, $\rho(t)$, especially in the case of a fast reactor. Consequently, in cases

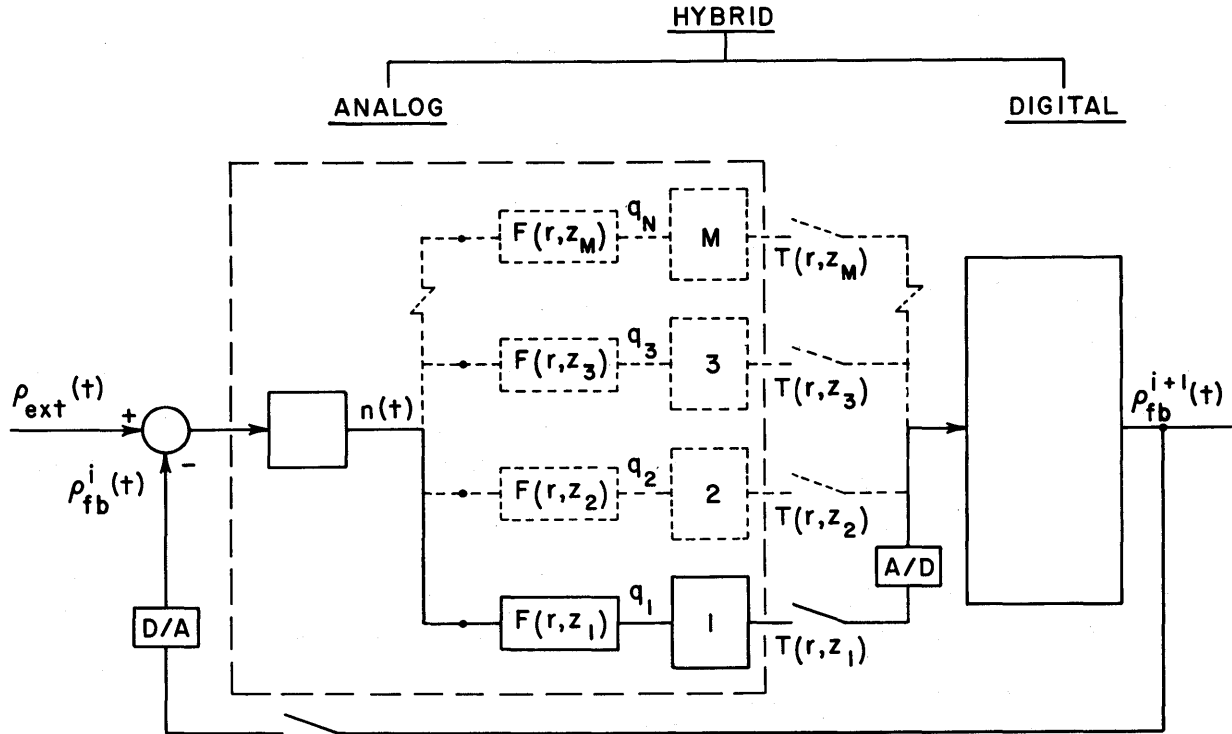


Figure 8. Schematic of the hybrid computer.

where the feedback is a strong function of temperature, a good initial guess of the feedback is necessary. This may be made in many cases by the following simple computational procedure. A closed loop is formed on the analog considering only one axial segment of an "average" fuel cell, and the reactivity feedback is obtained from the average temperatures of the various materials in this segment. This feedback reactivity is sampled and used as the initial guess. If the magnitude of the external reactivity, $|\rho_{ext}(t)|$, itself is small, an initial guess of ρ_{fb} to be zero is found to be satisfactory.

A typical example of the process of convergence is illustrated in Fig. 10, where the time-dependent part of the neutron flux $n(t)$ is plotted. It is this quantity whose variation in time is the most noticeable. The response corresponds to a ramp reactivity input $\rho_{ext}(t)$ which levels off after a second. The feedback is negative.

The converged neutron flux $n(t)$, the centerline temperature of a typical fuel cell, and the feedback reactivity are shown on Fig. 11.

An important aspect of the time dependence of temperature and feedback reactivity is that it is considerably "smoother" than that of $n(t)$. This fact is particularly significant because it influences the

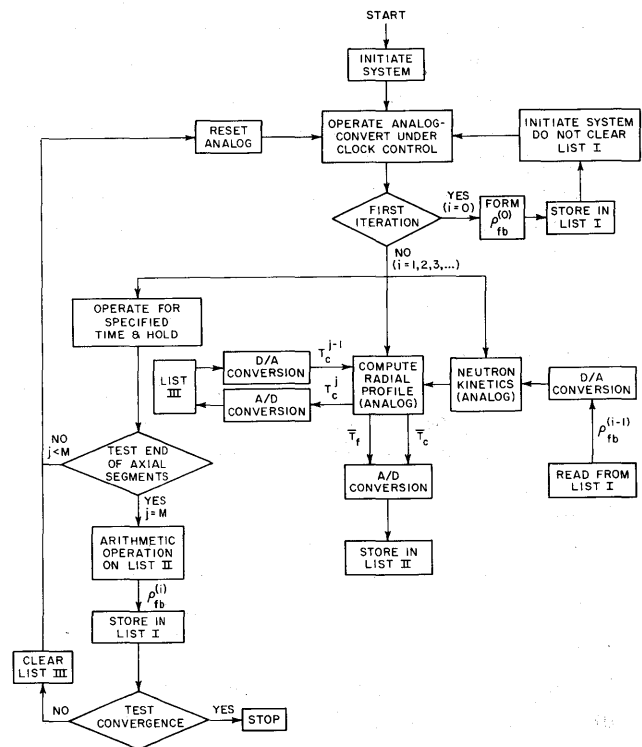


Figure 9. Flow chart.

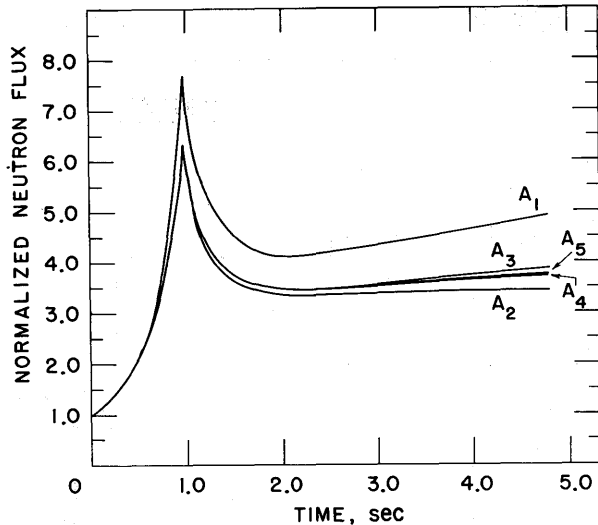


Figure 10. Illustration of convergence.

sampling rate (for A/D and D/A conversions as shown in Fig. 8) and the memory requirements of the digital computer.

Sampling rates were arrived at by means of the following experiment. A simple feedback loop (considering space independent temperatures) whose response is similar to that of the actual system is simulated on the analog. The closed loop response of the simple system is also obtained by the iterative procedure using progressively higher sampling rates and compared with the all-analog response. A sampling rate of 350/sec was found to be satisfactory for the problems considered.

In the present simulation, the iteration is terminated by the following test for the convergence of $\rho_{fb}(t)$.

$$\left| \rho_{fb}^{(i)}(t) - \rho_{fb}^{(i-1)} \right|_{\max} < \epsilon \quad (12)$$

A satisfactory value for ϵ depends upon the type of transient considered, and is largely a matter of the analyst's judgment. Five to six iterations were sufficient to achieve convergence in most cases.

Examples

The examples that follow represent typical problems solved using the hybrid simulation of the system.

Example 1. This example is presented to illustrate the efficiency of multiplexing in the computation of detailed temperature profiles.

Consider the transient conditions resulting from a decreasing coolant flow rate, say due to pump

failure; and the subsequent power shutdown of the reactor. The two independent forcing functions on the system now are: the decrease of coolant flow rate, represented by a decaying exponential, and the nature of the subsequent power shutdown (this happens after a certain interval of time determined by the design of the sensing devices). These forcing functions are given in Fig. 12.

The axial profile of the coolant temperature (see Fig. 12) under these circumstances is a highly complicated function of time. An accurate knowledge of this, however, is very necessary in order to ascertain the possibility of boiling of the coolant at any time during transients. This is particularly important in the case of a ceramic fuel whose temperature response is sluggish because of its low thermal diffusivity.

Example 2. The role of internal radiative heat transfer in ceramic fuels such as UO_2 is of current interest in the design of high-performance, fast-power breeder reactors. It is well established that internal radiation improves the thermal conductivity of the

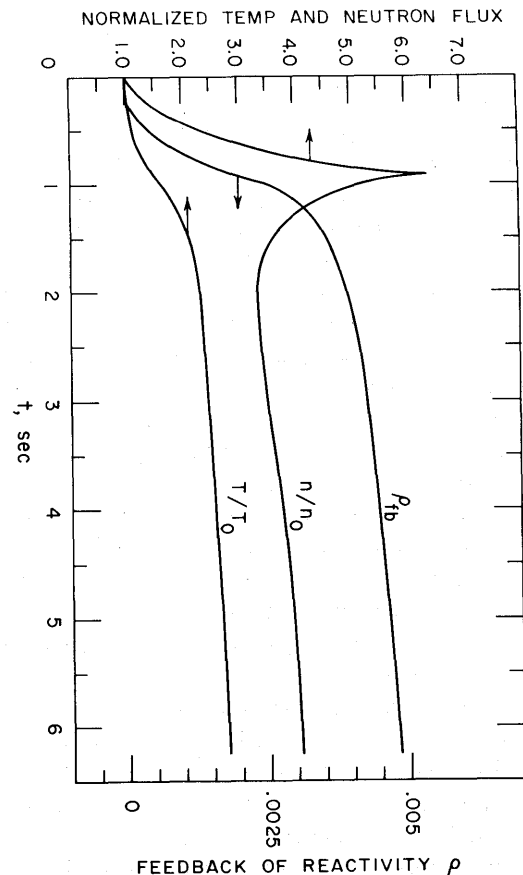


Figure 11. Converged variables.

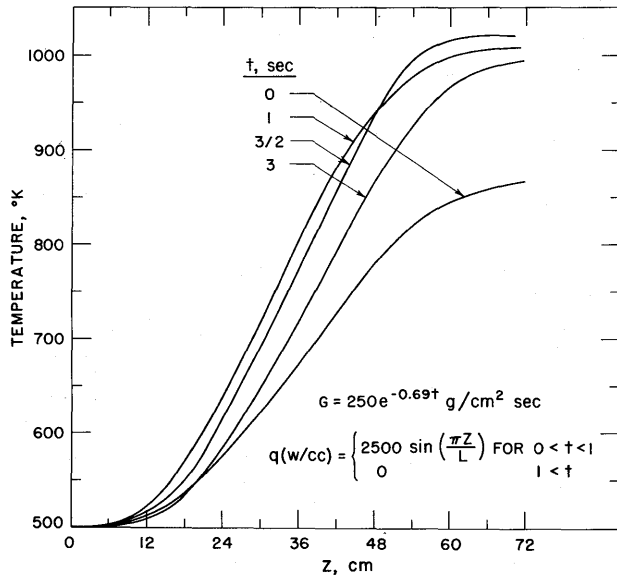


Figure 12. Axial profile of coolant temperature.

material at high temperatures. However, a parametric study of the reactor performance using materials having widely varying thermal properties⁷ leads to the solution of a set of quasi-linear parabolic partial differential equations for heat flow. No simple approach to the solution of these equations successfully estimates the influence of the *differences in the variation* of thermal properties upon the transient behavior of the overall reactor system. The present hybrid approach has been found very efficient for such problems.

Figure 13 illustrates the transient behavior of the maximum fuel temperature in a typical fuel cell when the reactor power increases. In the figure, Curve A gives the response when the contribution of internal radiation to thermal conductivity $K(T)$ is considered and Curve B when the term representing this contribution is artificially removed in the expression for the temperature dependence of $K(T)$. The dependence of K upon temperature for each of these two cases is shown in Fig. 14, by Curves K_1 and K_2 respectively.

Knowledge of the temperature profiles in the fuel cell is necessary to estimate thermal stresses and to determine whether fuel melting occurs during a power or coolant flow transient.

THE PRINCIPLE OF CONTRACTION MAPPING

The techniques employed to simulate the nuclear reactor system are applicable to a wide class of feed-

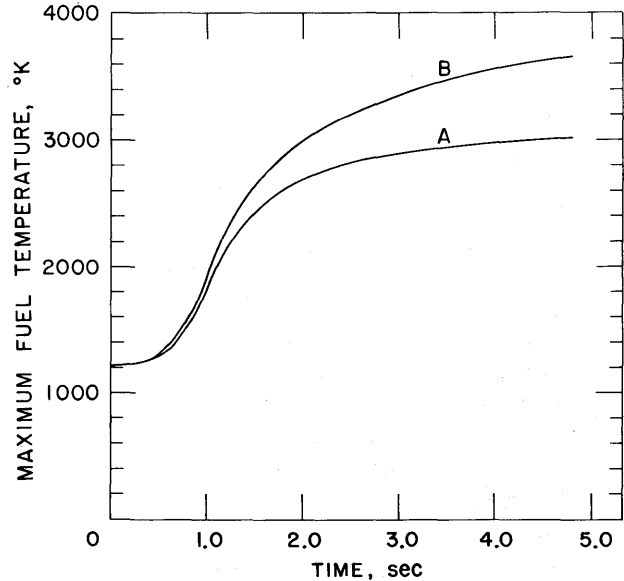


Figure 13. Effect of internal radiation on maximum fuel temperature.

back systems. The two important aspects of the present simulation are the use of the multiplexing technique and the consequent need for an iterative procedure.

The success of this method of analyzing the response of a closed loop system depends essentially upon the convergence of the iterative process. Also, a proof of convergence is necessary to provide mathematical rigor for the present method. It is found that such a proof is possible through the application of the principle of contraction mapping.

The system in Fig. 8 can be considered as one that "maps" (or transforms) $\rho_{fb}^{(i-1)}(t)$ to $\rho_{fb}^{(i)}(t)$.

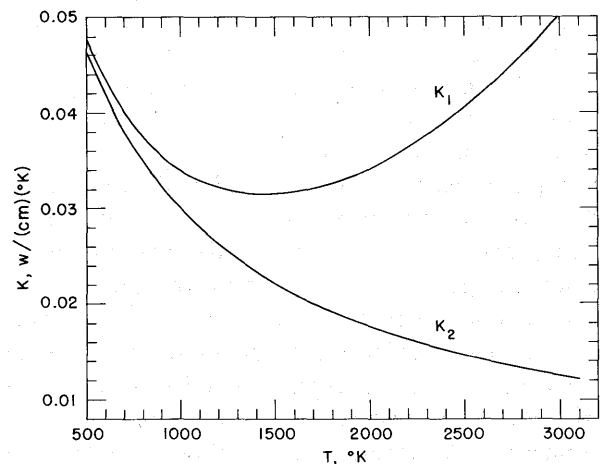


Figure 14. Thermal conductivity with and without internal radiation.

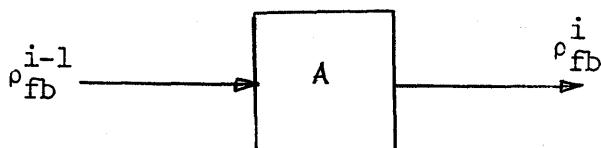


Figure 15. Conceptual representation of Fig. 8.

This is represented conceptually in Fig. 15. In the figure, A is an open loop operator.* For the iteration process to converge, the operator A has to satisfy certain conditions.

Consider two arbitrary functions of time, $x_1(t)$ and $x_2(t)$. Let the operator A map these two functions into $y_1(t)$ and $y_2(t)$, respectively (see Fig. 16a). Let $\mu(x_1, x_2)$ and $\mu(y_1, y_2)$ represent a measure of the "distance" between these functions. If A is such that

$$\mu(y_1, y_2) < \mu(x_1, x_2) \quad (13)$$

then the operator A produces a "contraction" in the "distance". Now suppose that A operates on $x_1(t)$ to produce $x_2(t)$, and in turn operates on $x_2(t)$ to produce $x_3(t)$ (see Fig. 16b). If A satisfies the condition in Eq. (13),

$$\mu(x_2, x_3) < \mu(x_1, x_2) \quad (14)$$

If this operation is iterated $(n-1)$ times,

$$\mu(x_n, x_{n-1}) < \mu(x_{n-1}, x_{n-2}) < \dots < \mu(x_1, x_2) \quad (15)$$

Equation (15) implies that there is a continuous reduction in the "distance" between the successive iterations. This is the concept of contraction mapping. The term "distance" between two functions is used in a general sense. Distance is a positive quantity and it satisfies the well-known triangle inequality. The norm, defined as the least upper bound of the absolute value of the difference between two functions, is a well known example of the distance.⁸

The convergence of the iterative process is closely related to the overall loop gain in the system. The term gain of an operator is defined as the least upper bound of the ratio between the norms, $\|A(x_i) - A(x_j)\|$ and $\|x_i - x_j\|$, where A denotes any operator and x_i and x_j any two functions of time.

In general, there are two types of convergence. The first one is called the geometric convergence, in which the rate of convergence is independent of the time interval for which the response is computed. Geometric convergence occurs only in systems in

* Note that this situation is a consequence of multiplexing.

which the loop gain is always less than unity. This is proved and discussed in Ref. 8. Physical systems with such a severe restriction on gain are seldom encountered in practice.

The second type of convergence is called uniform convergence. Here the rate of convergence, however, depends upon the time interval for which the response is required. Systems in which this type of convergence occur have an integration operator in the loop as shown in Fig. 17 or a pure delay. The gain of the rest of the operators A and B need only be finite in this case. It is noted that the integration may occur anywhere in the loop, and not necessarily between A and B . Proofs of convergence and the uniqueness of the converged response for the case of systems reducible to this general form are given by Zames.^{9, 10}

The following methodology for the proof of convergence of the iteration process is found to be efficient. First, an attempt is made to prove that the loop gain is always less than unity. This may be facilitated by some suitable mathematical transformations of the system variables. If the gain cannot be shown to be less than unity, as might frequently be, then the alternative is to prove that the system can be reduced to the general form in Fig. 17 or to show that there exists a pure time delay in the loop. Many times this may be accomplished directly or by transformation of variables. Since the transformed variables may sometimes not be physical quantities, particular care must be exercised in demonstrating the boundedness of the gains of the various operators present in the general form.

The reactor system under consideration can be transformed into the general form shown in Fig. 17. A proof of this statement is given in detail in Appendix 1.

It follows, therefore, that the convergence of the iterative process here is of the second type. Consequently, the number of iterations necessary increases with the time interval for which the response is computed. This may be observed in Fig. 10. When the

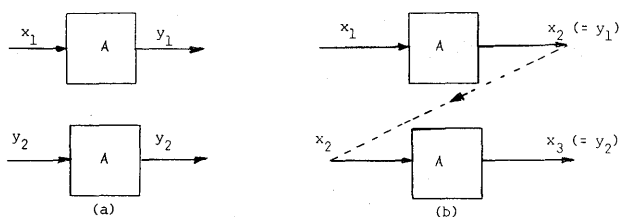


Figure 16. Illustration of contraction mapping.

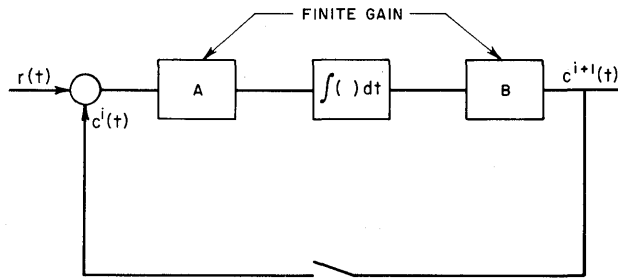


Figure 17. Generalized form of the reactor system.

interval is large, a correspondingly large amount of memory will be required to store the sampled variables. A sizeable economy in the memory units may be effected if the response is computed sequentially by subdividing the time interval into two or more smaller intervals and initializing the system variables in any of these intervals with their final values of the preceding interval.

DISCUSSION

The outstanding feature of the present method of analyzing the response of a closed loop system is the employment of an iterative procedure. This allows multiplexing of computer components.

As a result of multiplexing, there is a large saving in analog circuitry. But for this economy, it would not have been possible to employ the (limited amount of) analog components to handle the enormous number of differential equations in the mathematical model of the reactor core. It is noted that the simulation of these nonlinear equations on the analog results in better accuracy than is possible with existing codes for pure digital computers.*

The success of the present method of analyzing the response of any closed loop system depends essentially upon the convergence of the iterative process. From a practical standpoint a mathematical proof of convergence alone is not enough. Simulation of typical transients is necessary to obtain a quantitative estimate of the necessary number of iterations.

The iterative process has facilitated simulation in continuous time for rather large intervals of time.† The length of the time interval is limited only by

* Codes such as FORE, TER 4, and ARGUS give less accurate results when the parameters (thermal properties) in the heat flow equations are strongly dependent upon temperature. An effort is presently underway to find better differencing schemes to deal with the nonlinearities of these parabolic partial differential equations.

† See Ref. 3 for a simulation in which it is necessary to integrate differential equations by means of the analog for small time steps.

practical considerations such as the available memory, required sampling rate and necessary number of iterations for convergence. If the response need be computed for a long interval, this interval may be subdivided into smaller intervals (whose length is compatible with the practical limitations) and the response computed sequentially. As discussed in the preceding section, in cases where convergence depends upon the response time, a large saving in computing time and memory units is possible by subdividing the response time.

One practical aspect of multiplexing of analog circuitry is the frequent re-initialization of the variables and modification of the parameters in the equations. This is accomplished by changing the various potentiometer settings. From experience, it is found that the incorporation of an automatic pot setting equipment is of great help. In a completely automated set up it is suggested that the digital computer be programmed to effect the necessary changes in pot settings.

Very few arithmetic operations are done *during* sampling intervals in the present simulation. Therefore, the speed of the digital computer is not a limiting factor. Consequently, high-speed analog equipment may be used to advantage.

The hybrid simulation is proving itself to be effective and economical in the analysis of transience in normal nuclear reactor operations and in hypothetical accident conditions. It can be particularly useful in the design studies of fast reactor cores composed of ceramic fuels whose thermal properties vary significantly with temperature.

The techniques used in the hybrid simulation are general enough to be applicable to any feedback system in which the iteration is a contraction mapping. The advantages of the techniques increase with the amount of multiplexing and the degree of complexity in the equations. It is therefore recommended that further effort be made to find applications of this method of simulation to other mathematical models composed of nonlinear partial differential equations.

APPENDIX 1

MATHEMATICAL PROOFS

I: To prove that the overall reactor system representation in Fig. 6 can be reduced to the form in Fig. 17.

The neutron kinetics equations with one set of delayed neutrons is considered in this section.

$$\frac{dn}{dt} = \frac{1}{l} [\rho(t) - \beta] n + \lambda c \quad (16)$$

$$\frac{dc}{dt} = \frac{\beta}{l} n - \lambda c$$

Let $n = n_0$ and $c = c_0$ at $t = 0$. Also, let the reactor be critical (i.e., $\frac{dn}{dt} = \frac{dc}{dt} = 0$) initially so that

$$\frac{n_0}{c_0} = \frac{\lambda l}{\beta}$$

Define:

$$N \equiv \frac{n}{n_0} \quad (17)$$

$$C \equiv \frac{c}{c_0}$$

Substituting (17) in (16),

$$\frac{dN}{dt} = \frac{1}{l} [\rho(t) - \beta] N + \left(\frac{\beta}{l}\right) C \quad (18)$$

$$\frac{dC}{dt} = \lambda N - \lambda C$$

at $t = 0$, $N = 1$, and $C = 1$.

Define a second set of new variables:

$$u \equiv \ln N = \ln \frac{n}{n_0} \quad (19)$$

and

$$v \equiv \frac{C}{N}$$

Substituting (19) in (18),

$$\frac{du}{dt} = \frac{1}{l} [\rho(t) - \beta] + \frac{\beta}{l} v \quad (20)$$

$$\frac{dv}{dt} = \lambda - \lambda v - v \frac{du}{dt}$$

Substituting for du/dt from Eq. (20),

$$\frac{dv}{dt} = \lambda - \left\{ \lambda + \frac{1}{l} [\rho(t) - \beta] \right\} v - \frac{\beta}{l} v^2 \quad (21)$$

Equations (20) and (21) can be represented in the block diagram in Fig. 18. Observe that Eq. (20)

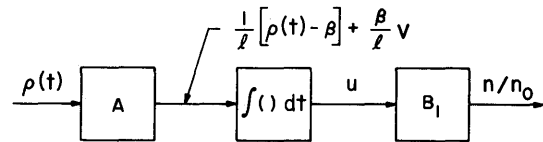


Figure 18. Block diagram of Eqs. (20) and (21).

represents a pure integration, since the right-hand side does not involve u .

B_1 is the operation e^u .

Two more blocks B_2 and B_3 representing the heat flow equations and the feedback are added to obtain the overall loop. Figure 19 is equivalent to Fig. 17 if B_1, B_2, B_3 are combined to form B .

The iterative process would converge only if the gain of A, B_1, B_2 , and B_3 are finite. It is, therefore, necessary to show that if the inputs of A, B_1, B_2 , and B_3 are finite, their outputs would remain bounded.

II: To show that the gain of A is bounded. For this it is necessary to show that $v(t)$, given by Eq. (21), is bounded if $\rho(t)$ is bounded.

Equation (21) may be written as

$$\frac{dv}{dt} = \lambda - \zeta(t)v - \frac{\beta}{l} v^2 \quad (22)$$

where $\zeta(t) \equiv \lambda + \frac{1}{l} [\rho(t) - \beta]$, and $v(0) = 1$.

Equation (22) is in the form of the famous Riccati equation. Let the maximum absolute value of $\zeta(t)$,

$$|\zeta(t)|_{\max} \equiv \zeta_{\max}$$

The right-hand side of Eq. (22),

$$\lambda - \zeta(t)v - \frac{\beta}{l} v^2 \leq \lambda + \zeta_{\max} v - \frac{\beta}{l} v^2 \quad \text{for all } t$$

Let v_{\max} be defined as the positive value of v for which

$$\lambda + \zeta_{\max} v - \frac{\beta}{l} v^2 = 0$$

I.e.,

$$v_{\max} \equiv \frac{\zeta_{\max} + \sqrt{\zeta_{\max}^2 + \frac{4\beta\lambda}{l}}}{2\frac{\beta}{l}} \quad (23)$$

Observation 1: If the initial value $v(0)$ (namely 1) exceeds v_{\max} , the right-hand side of Eq. (22) is negative, i.e., dv/dt is negative. Therefore v cannot exceed the initial value, v_{\max} .

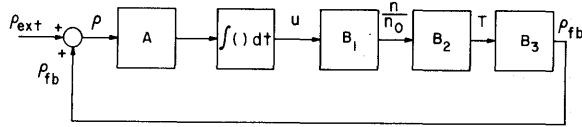


Figure 19. Block diagram of transformed reactor system.

If $v(0) < v_{\max}$, the right-hand side of Eq. (22) may be positive or negative. But in any case, the corresponding $v(t)$ cannot exceed v_{\max} , because dv/dt is negative for $v(t) > v_{\max}$.

Observation 2: v shall never be less than zero because when $v = 0$, dv/dt of Eq. (22) is equal to λ , which is positive, and consequently, v cannot decrease any further.

From observations 1 and 2 it is obvious that v remains bounded between 0 and 1 if $v_{\max} < 1$, and between 0 and v_{\max} if $v_{\max} > 1$.

III: To prove that the gain of A is bounded in the more general case of more than one group of delayed neutrons.

The kinetics equations now take the following form:

$$\begin{aligned} \frac{dn}{dt} &= \frac{1}{l} [\rho(t) - \beta]n + \sum_{i=1}^J \lambda_i c_i \\ \frac{dc_i}{dt} &= \frac{\beta_i}{l} n - \lambda_i c_i \quad i = 1 \dots J \end{aligned} \quad (24)$$

where $\beta = \sum_{i=1}^J \beta_i$, and J is usually 6.

The substitutions (25) and (26) are made successively in Eq. (24) to obtain Eqs. (27) and (28) respectively.

$$N \equiv \frac{n}{n_0} \quad (25)$$

and

$$C_i \equiv \frac{c_i}{c_{i0}}$$

$$u \equiv \ln N \quad (26)$$

$$v_i \equiv \frac{C_i}{N}$$

$$\frac{du}{dt} = \frac{1}{l} [\rho(t) - \beta] + \sum_{i=1}^J \frac{\beta_i}{l} v_i \quad (27)$$

$$\frac{dv_i}{dt} = \lambda_i - \lambda_i v_i - v_i \frac{du}{dt}$$

Substituting for du/dt given by Eq. (27),

$$\frac{dv_i}{dt} = \lambda_i - [\lambda_i + \frac{1}{l}(\rho(t) - \beta)] v_i - v_i \left\{ \sum_{k=1}^J \frac{\beta_k}{l} v_k \right\} \quad (28)$$

$i = 1 \text{ to } J$

The boundedness of the gain of A is shown by proving that v_i obtained by solving Eq. (28) will remain bounded if $\rho(t)$ is bounded. The proof is similar to that in II.

Equation (28) may be written as

$$\frac{dv_i}{dt} = \lambda_i - \zeta_i(t) v_i - v_i \left\{ \sum_{k=1}^J \frac{\beta_k}{l} v_k \right\} \quad (29)$$

where $\zeta_i(t) \equiv \lambda_i + \frac{1}{l} [\rho(t) - \beta]$

$v_i(0) = 1$, and

$i = 1, \dots, J$.

Observe that $v_i(t)$ cannot become less than zero (for all i) because, when $v_i(t) = 0$, $\frac{dv_i}{dt} = \lambda_i$, which is positive.

$\therefore v_i(t) > 0$ for all i

Let the maximum absolute value of $\zeta_i(t)$ for all i be ζ_{\max} . I.e.,

$$|\zeta_i(t)|_{\max} \equiv \zeta_{\max}$$

The right-hand side of Eq. (28),

$$\lambda_i - \zeta_i(t) v_i - v_i \left\{ \sum_{k=1}^J \frac{\beta_k}{l} v_k \right\} \quad (30)$$

$$\leq \lambda_i + \zeta_{\max} v_i - \frac{\beta_i}{l} v_i^2$$

Let $v_{i \max}$ be defined as the positive value of v_i for which

$$\lambda_i + \zeta_{\max} v_i - \frac{\beta_i}{l} v_i^2 = 0 \quad (31)$$

Following the arguments in Observation 1 of II, it is clear that $v_i(t)$ remains bounded between 0 and 1 if $v_{i \max} < 1$ and between 0 and $v_{i \max}$ if $v_{i \max} > 1$.

IV: To prove that the gain of B_1 , B_2 and B_3 are finite.

B_1 is the operation e^u . Therefore, it is obvious that if u is bounded, e^u is bounded.

B_2 is represented by the heat flow equations.

By definition, the shape factor of neutron flux, $\psi(r)$, the fission cross-section distribution, and energy release per fission are all bounded functions.

Therefore, if $\frac{n}{n_0}$, the input to B_2 , is bounded the heat generation rate in the heat flow equations would remain bounded.

Using the proofs given in Ref. 11 for the existence of the solution of quasi-linear parabolic partial differential equations (equations of heat flow) it is shown that T remains bounded for any arbitrary length of time.

Therefore, the gain of B_2 is bounded.

The algebraic expressions that give the feedbacks as a function of T are well behaved. See Eqs. (4) and (5). I.e., given that T , the input of B_3 , is bounded the output ρ_{fb} will be bounded. Hence, the gain of B_3 is bounded.

APPENDIX 2

DISCUSSION OF THE MATHEMATICAL MODEL

Neutron Kinetics Equations

The derivation of the neutron kinetics equations from the time-dependent transport equation has been treated in detail by A. F. Henry¹² and E. P. Gyftopoulos.¹³ The neutron flux F in general is considered to be space, r ; time, t ; energy, E ; and direction, Ω dependent. Without loss of generality F may be expressed as:

$$F(r, E, \Omega, t) = \psi(r, E, \Omega, t) n(t) \quad (32)$$

where $n(t)$ is a time-dependent function accounting for all growth or decay tendencies of the neutron density while $\psi(r, E, \Omega, t)$ is a shape function, which is bounded at all times.

Several methods have been proposed to obtain the shape function $\psi(r, E, \Omega, t)$. A simultaneous solution of $n(t)$ and $\psi(r, E, \Omega, t)$ has not been successful to date. However, some of the quasi-static models¹² seem to be computationally feasible. The quasi-static models essentially neglect the time derivative of ψ in the computation of $n(t)$. This allows the computation of $n(t)$ independent of ψ .^{12, 13} ψ is computed periodically during transience. Consequently, the time dependence of ψ is discretized and is implicit in the changing nuclear properties and physical dimensions.

In the present analysis, the flux shape, ψ , is taken as invariant during transience. Work is in progress to consider the more realistic case in which ψ does vary with power level. Consequently, the present treatment of the neutronics is macroscopic in nature. The neutron flux is represented by the product $\psi(r) n(t)$; in which the angular and energy dependences of ψ are integrated over all Ω and E . The equations to solve now reduce to the set of ordinary differential equations in Eq. (2).

Transient Heat Flow Equations

The heat flow equations are written for a cylindrical fuel cell, shown in Fig. 4. The following are the assumptions.

The internal heat generation rate is symmetric with respect to the longitudinal axis of the cell, and the materials are isotropic.

The axial conduction of heat and the radial variation of the coolant temperature are neglected.

Materials do not have phase changes (fuel melting and coolant boiling) during transience. Further, there is no large scale disassembly of the core.

Under these assumptions, the heat conduction equations are written in one-dimensional space; namely, r , at any position z along the axis.

$$c\rho r \frac{\partial T}{\partial t} = \frac{\partial}{\partial r} \left[K(T) r \frac{\partial T}{\partial r} \right] + r q(r, z, t) \quad (\text{for fuel}) \quad (33)$$

$$c\rho r \frac{\partial T}{\partial t} = \frac{\partial}{\partial r} \left[K(T) r \frac{\partial T}{\partial r} \right] \quad (\text{bond or clad}) \quad (34)$$

where T = the temperature,

c = the specific heat,

ρ = the density,

K = the thermal conductivity, and

q = the internal volumetric heat generation rate.

$q(r, z, t)$ is expressed as:

$$q(r, z, t) = F(r, z) n(t) \quad (35)$$

where $F(r, z)$ is a conversion factor to obtain the fission heat generation rate from the neutron flux density.

The boundary conditions are:

$$\left. \frac{\partial T}{\partial r} \right|_{r=0} = 0 \quad (\text{assuming the material to be isotropic}) \quad (36)$$

$$-K(T) \left. \frac{\partial T}{\partial r} \right|_{r=R} = h(T_R - T_c) \equiv Q \quad (37)$$

where T_R is the temperature at the surface of the clad, and T_c the average coolant temperature.

The surface heat transfer coefficient h is a function of temperature and coolant flow rate. Q represents the heat transferred per second per unit area of the boundary between the fuel cladding and the coolant.

The integration of Eqs. (33) or (34) in each radial increment Δr results in Eq. (7). The temperature between any two node points is assumed to vary linearly. Note that this assumption is fully preserved in the integration process.

The rise coolant temperature for any axial increment may be obtained by writing the heat balance equation. This equation in its differential form may be stated as:

$$\frac{\partial}{\partial t} [\rho c T_c] + \frac{\partial}{\partial z} [G c T_c] = \wedge Q \quad (38)$$

where ρ = the density,

c = the specific heat,

T_c = the temperature of the coolant,

G = the mass flow rate per unit cross-sectional area of coolant, and

\wedge = the ratio between the circumference of the clad and the cross-sectional area of coolant flow.

If ρ and c are constants,* Eq. (38) may be rewritten as:

$$\frac{\partial T_c}{\partial t} + \frac{G(t)}{\rho} \frac{\partial T_c}{\partial z} = \frac{\wedge}{\rho c} Q \quad (39)$$

Equation (39) is integrated along Δz , for the i th increment of the z -axis, to obtain Eq. (10).

ACKNOWLEDGMENTS

The authors wish to thank Professor J. J. Levin of the University of Wisconsin and Dr. Isaac Klinger of Argonne National Laboratory for helpful discussions on the principle of contraction mapping.

* If ρ and c are functions of temperature, Eq. (38) has to be solved simultaneously with the mass balance equation.

REFERENCES

1. G. and T. M. Korn, *Electronic Analog and Hybrid Computers*, McGraw-Hill, New York, 1964.
2. R. M. Howe, "Hybrid Solution of Partial Differential Equations," University of Michigan Engineering Summer Conferences, Hybrid Computation, July 1965.
3. R. Ruzskay and E. E. L. Mitchell, "Hybrid Simulation of a Reacting Distillation Column," *1966 Spring Joint Computer Conference*, Spartan Books, Washington, D.C., pp. 389-99.
4. A. D. Feith, "The Thermal Conductivity of UO_2 up to $2500^\circ C$," *J. of Nuclear Materials*, vol. 16, p. 231 (1965).
5. J. Belle (ed.), "Uranium Dioxide: Properties and Nuclear Applications, Naval Reactors," Division of Reactor Development, U.S. Atomic Energy Commission, July 1961.
6. R. Viskanta, "Influence of Internal Thermal Radiations on Heat Transfer in UO_2 Fuel Elements," *Nucl. Sci. and Engineering*, vol. 21, p. 13 (1965).
7. C. K. Sanathanan et al, "Transient Temperature Distributions in Fast Reactor Fuels with Widely Varying Thermal Diffusivity," International Conference on Fast Reactors, Argonne National Laboratory, Oct. 1965.
8. A. N. Kolmogorov and S. V. Fomin, *Elements of the Theory of Functions and Functional Analysis*, Vol. 1, 1st Russian ed. 1954, translated by Leo F. Boron, Graylock Press, Rochester, N. Y., 1957.
9. G. Zames, "Functional Analysis Applied to Nonlinear Feedback Systems," *IEEE Transactions on Circuit Theory*, Sept., 1963, p. 392.
10. —, "Nonlinear Operators for System Analysis," MIT Research Laboratory of Electronics, Technical Report 370 (Aug. 1960).
11. A. F. Filippov, "Conditions for the Existence of Solution for a Quasi-Linear Parabolic Equation," *Soviet Mathematics, DOKLADY*, vol. 2, p. 1517 (1961).
12. A. F. Henry, "The Application of Reactor Kinetics to the Analysis of Experiments," *Nucl. Sci. and Engineering*, vol. 3, pp. 52-70 (1958).
13. E. P. Gyftopoulos, "Point Reactor Kinetics and Stability Criteria," Third United Nations International Conference on the Peaceful Uses of Atomic Energy, Geneva, P/270 (May 1964).

A HYBRID COMPUTER SOLUTION OF THE CO-CURRENT FLOW HEAT EXCHANGER STURM-LIOUVILLE PROBLEM *

Lawrence T. Bryant, Lawrence W. Amiot and Ralph P. Stein

*Argonne National Laboratory
Argonne, Illinois*

INTRODUCTION

Analog and digital computers have proven to be very valuable tools for solving engineering problems. In order to obtain solutions to some of the problems, trial-and-error or searching techniques must be employed. Two-point boundary value problems—of which the classical Sturm-Liouville system is a special but important case—belong to that class of problems.

The analog computer is particularly suited for solving linear or nonlinear differential equations. However, those problems which require trial-and-error techniques to effect a solution are often quite time-consuming, particularly if a large number of solutions is required. This is compounded if a large number of arithmetic operations must also be performed, since the accuracy of an analog computer when performing a large number of such operations may not be ideal. Hence, solution by pure analog computation of problems of this kind can be quite expensive and can introduce questions concerning the overall accuracy of the results obtained.

The searching or trial-and-error technique can be done more economically by programming the procedure on a digital computer. Also, the arithmetic op-

erations can be performed with relatively unlimited accuracy at very little cost. The solution of the differential equations, however, can be more time-consuming on a digital computer when compared with the speed at which they can be solved on an analog computer, especially when a large number of different cases must be considered. Further, the programming necessary to solve the equations by digital computation will often be more expensive under the same comparison. The use of a hybrid computer, which employs both analog and digital elements, provides a means by which the aforementioned difficulties may be overcome.

This paper presents the solution by hybrid computation of the boundary value problem which results from an analysis^{1,2} of the co-current laminar-flow double-pipe heat exchanger. The analysis leads to a Sturm-Liouville system consisting of two Sturm-Liouville differential equations coupled at a common boundary. In addition to the use of an eigenvalue searching procedure, which must be handled by the computer operator, solution by analog computation requires a much larger number of arithmetical operations than with the more familiar two-point boundary value problem.

A hybrid computer can perform all of the operations necessary to solve problems of this type automatically and economically. The digital elements of

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

the hybrid computer perform the arithmetic, trial-and-error, and control operations, as well as input/output, while the analog elements provide the solutions of the Sturm-Liouville differential equations and the generation of subsidiary functions. Thus, the entire computing operation is handled by the machine, relieving the computer operator of the tedious job of "searching," as well as performing the arithmetic operations with improved accuracy.

Formulation of the problem and the method devised to effect the solutions are described. Typical results are shown in tabular form; maximum errors vary, but generally are less than 0.2%.

THE MATHEMATICS OF THE CO-CURRENT FLOW HEAT EXCHANGER

The usual co-current flow double-pipe heat exchanger consists of two concentric pipes with fluids flowing in parallel through the annular space and central tube as illustrated in Fig. 1. The figure also shows some of the nomenclature to be used below.

Energy conservation, with appropriate simplification, together with Fourier's heat conduction law leads to partial differential equations for each channel of the exchanger¹⁻³ with boundary and initial (or inlet) conditions. The resulting mathematical problem, which determines the temperature distributions $\xi_i(x,z)$, $i=1,2$ as functions of the dimensionless

space variables x and z (see Fig. 1), can be written as follows:

$$\frac{1}{x} \frac{\partial}{\partial x} \left(x \frac{\partial \xi_1}{\partial x} \right) = g_1(x) \frac{\partial \xi_1}{\partial z} \tag{1a}$$

$$\frac{1}{1-(1-R)x} \frac{\partial}{\partial x} \times \left\{ [1-(1-R)x] \frac{\partial \xi_2}{\partial x} \right\} = \omega^2 g_2(x) \frac{\partial \xi_2}{\partial z} \tag{1b}$$

with $\xi_i(x,z)$, $0 \leq x \leq 1$, $0 \leq z \leq \infty$, and

$$\omega^2 = \frac{R}{1+R} KH.$$

Inlet conditions

$$\xi_1(x,0) = 0; \xi_2(x,0) = 1 \tag{2a,b}$$

Boundary conditions

$$\left. \frac{\partial \xi_i}{\partial x} \right|_0 = 0 \quad i = 1,2 \tag{3a,b}$$

$$K \left. \frac{\partial \xi_1}{\partial x} \right|_1 + \left. \frac{\partial \xi_2}{\partial x} \right|_1 = 0 \tag{3c}$$

$$K_w \left. \frac{\partial \xi_1}{\partial x} \right|_1 + \xi_1(1,z) - \xi_2(1,z) = 0 \tag{3d}$$

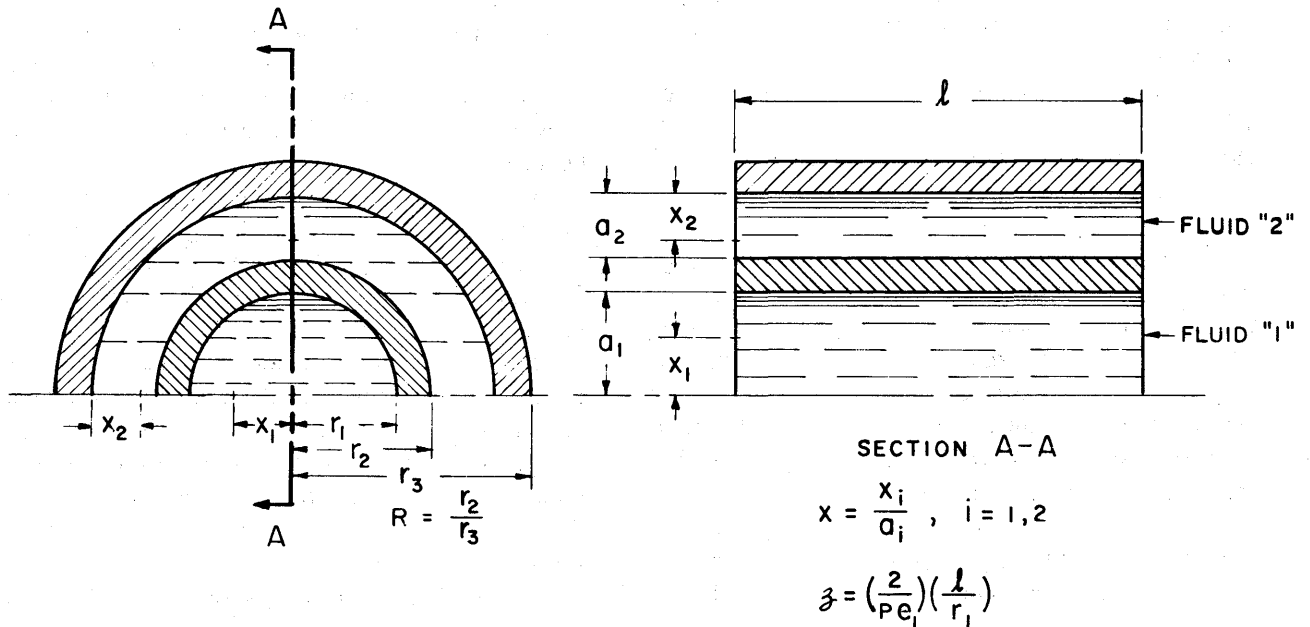


Figure 1. The double-pipe heat exchanger.

The quantities H , K , and K_w are dimensionless parameters which, along with the annulus radius ratio R (see Fig. 1), define the geometry and operating conditions of the heat exchanger. The parameter H is the heat capacity mass flow rate ratio of the two fluids; K is a relative thermal resistance for heat flow from the fluid in the annular space; and K_w is the relative thermal resistance of the tube wall separating the two fluids. The parameters R , H , and K_w are independent of each other; the parameter K , however, depends on R and the ratio of the fluid thermal conductivities γ , viz:

$$K = \frac{(1-R)\gamma}{R}$$

Thus, except for the limiting case of a "narrow" annular space ($R \rightarrow 1$), the exchanger geometry and operating conditions are best defined by the four independent parameters R , H , γ , and K_w . For the limiting case of a narrow annular space, K is used in place of γ , and R is eliminated as a separate parameter. Since further knowledge of the mathematical definitions of these parameters is not important to the purposes of this paper, they will not be given here. Instead, the reader interested in more than the specific topic of computation discussed here is directed to the appropriate references.¹⁻³

The functions $g_i(x)$, $i=1,2$, represent the "fully developed" local fluid velocity divided by the average velocity. For laminar flow through a tube

$$g_1(x) = 2(1-x^2) \quad (4a)$$

while for laminar flow through an annular space,

$$g_2(x) = \frac{2[1-r(x) + c \ln r(x)]}{1 + R^2 - 2c} \quad (4b)$$

with

$$r(x) = [1 - (1-R)x]^2 \quad (4c)$$

and

$$c = -\frac{1-R^2}{4 \ln R} \quad (4d)$$

These expressions for $g_i(x)$ or their equivalents can be found in most textbooks on fluid mechanics.

As discussed in Refs. 1-3, separation of variables applied to Eqs. (1) to (3) leads to solutions of the form

$$\xi_i(x,z) = \frac{H}{1+H} + \sum_{n=1}^{\infty} c_n E_{i,n}(x) e^{-\lambda_n^2 z} \quad i=1,2 \quad (5a,b)$$

where the c_n are generalized Fourier coefficients determined so that Eqs. (5) satisfy the inlet conditions given by Eqs. (2), and the $E_{i,n}(x)$ are eigenfunctions corresponding to eigenvalues λ_n^2 . The eigenfunctions, which allow Eqs. (5) to satisfy the boundary conditions given by Eqs. (3), are defined by the following "two-region" Sturm-Liouville system.

$$\frac{d}{dx} [x E'_{1,n}] + x g_1 \lambda_n^2 E_{1,n} = 0 \quad (6a)$$

$$\frac{d}{dx} \{ [1 - (1-R)x] E'_{2,n} \} + [1 - (1-R)x] g_2 \omega^2 \lambda_n^2 E_{2,n} = 0 \quad (6b)$$

$$E'_{i,n}(0) = 0 \quad i=1,2 \quad (7a,b)$$

$$K E'_{1,n}(1) + E'_{2,n}(1) = 0 \quad (7c)$$

$$K_w E'_{1,n}(1) + E_{1,n}(1) - E_{2,n}(1) = 0 \quad (7d)$$

For reasons which will be apparent later, it is convenient to define the functions

$$G_1(x) = x g_1(x) \quad (8a)$$

and

$$G_2(x) = [1 - (1-R)x] g_2(x) \quad (8b)$$

The Fourier coefficients c_n are then given by

$$c_n = -\frac{B_{1,n}}{N_n} \quad (9)$$

where

$$B_{1,n} = \int_0^1 G_2(x) E_{1,n} dx \quad (10a)$$

$$= -2 E'_{1,n}(1) / \lambda_n^2 \quad (10b)$$

and

$$N_n = 2 \int_0^1 \left(G_1 E_{1,n}^2 + \frac{H}{1+R} G_2 E_{2,n}^2 \right) dx \quad (11)$$

The above system of equations and formulas is analogous to the more familiar "single-region" Sturm-Liouville problem. For example, the eigenvalues λ_n^2 $n=1,2,3, \dots$ are positive and denumerably infinite in number with $\lambda_n^2 < \lambda_{n+1}^2$; and an orthogonality condition for the $E_{i,n}(x)$ leads to Eq. (9).¹⁻³

For practical applications, the most important information desired from the mathematical solution is the relationship between the overall heat transfer rate and the exchanger heat transfer area as determined by the parameters R , H , γ , and K_w . This re-

relationship is conveniently expressed as a heat exchanger efficiency ε ($0 \leq \varepsilon \leq 1$) as a function of the dimensionless heat exchanger length z . Use of the mathematical solution given above results in¹⁻³

$$\varepsilon = 1 - \frac{1+H}{H} \sum_{n=1}^{\infty} \frac{B_{1,n}^2}{N_n} e^{-\lambda_n^2 z} \quad (12a)$$

which can be written as

$$\varepsilon = 1 - \sum_{n=1}^{\infty} \phi_n e^{-\lambda_n^2 z} \quad (12b)$$

with

$$\phi_n = \frac{1+H}{H} \frac{B_{1,n}^2}{N_n} \quad (13)$$

Thus, the most important quantities to determine are ϕ_n and λ_n^2 as functions of R , H , γ , and K_w ; or, for the limiting case of $R \rightarrow 1$, as functions of H , K , and K_w .

BASIS FOR COMPUTER SOLUTION

Let $y_i(x, \lambda)$, $i=1,2$, be the solution of Eqs. (6) which satisfy the "initial" conditions

$$y_i(0, \lambda) = 1 \quad (14a)$$

and

$$y_{i,x}(0, \lambda) = 0 \quad (14b)$$

for arbitrary values of λ . Let

$$F(x, \lambda) = y_1(x, \lambda) y_{2,x}(x, \lambda) + K y_{1,x}(x, \lambda) y_2(x, \lambda) + K_w y_{1,x}(x, \lambda) y_{2,x}(x, \lambda) \quad (15)$$

Then it is easily shown¹ that the eigenvalues λ_n^2 are obtained from the positive nonzero roots of $F(1, \lambda) = 0$, and that the eigenfunctions can be expressed as

$$E_{1,n}(x) = y_{2,x}(1, \lambda_n) y_1(x, \lambda_n) \quad (16a)$$

and

$$E_{2,n}(x) = -K y_{1,x}(1, \lambda_n) y_2(x, \lambda_n) \quad (16b)$$

Substitution of Eqs. (16) into Eqs. (10b) and (11) gives expressions for $B_{1,n}$ and N_n . These expressions are then substituted into Eq. (13) to give the following relationship for the desired quantities ϕ_n .

$$\phi_n = \frac{1+H}{H} \frac{2}{\lambda_n^4} \left[\frac{M_1(\lambda_n)}{y_{2,x}^2(1, \lambda)} + \frac{\omega^2 K}{R} \frac{M_2(\lambda_n)}{y_{1,x}^2(1, \lambda_n)} \right]^{-1} \quad (17)$$

where

$$M_1(\lambda) = \int_0^1 G_1(x) y_1^2(x, \lambda) dx \quad (18a)$$

and

$$M_2(\lambda) = \int_0^1 G_2(x) y_2^2(x, \lambda) dx \quad (18b)$$

The analog portion of the hybrid computer is wired to generate $y_i(x, \lambda)$, $i=1,2$, for arbitrary positive values of λ by simultaneous solution of Eqs. (6) over a time interval equivalent to $0 \leq x \leq 1$. The integrals $M_i(\lambda)$ are also computed and, along with values of $y_i(1, \lambda)$ and $y_{i,x}(1, \lambda)$, appear as output voltages at the end of the time interval.

Solution of Eqs. (6) and evaluation of the integrals $M_i(\lambda)$ require the analog to generate the functions $G_i(x)$, $i=1,2$. These functions are obtained by analog solutions of appropriate differential equations for $G_1(x)$ and $g_2(x)$ obtained by successive differentiation of Eqs. (8a) and (4). Thus, $G_1(x)$ is generated by solution of

$$G_1'''(x) = -12$$

with the initial conditions: $G_1''(0) = 0$, $G_1'(0) = 2$, and $G_1(0) = 0$.

The function $g_2(x)$ is generated by solution of

$$p''(x) = I_3(R)$$

and

$$[1 - (1-R)x] \frac{dg_2}{dx} = p(x)$$

with the initial conditions: $p'(0) = I_2(R)$, $p(0) = I_1(R)$, and $g_2(0) = 0$, where

$$I_3 = \frac{8(1-R)^3}{1+R^2-2c}$$

$$I_2 = -12 \frac{2(1-R)^2}{3(1+R^2-2c)}$$

$$I_1 = 6 \frac{2(1-R)(1-c)}{3(1+R^2-2c)}$$

with c given by Eq. (4d). For the limiting case of $R \rightarrow 1$; $I_3 = 0$, $I_2 = -12$, and $I_1 = 6$.

SOLUTION BY HYBRID COMPUTER

Figure 2 shows a flow diagram of the hybrid system. The hybrid computer employed consists of a

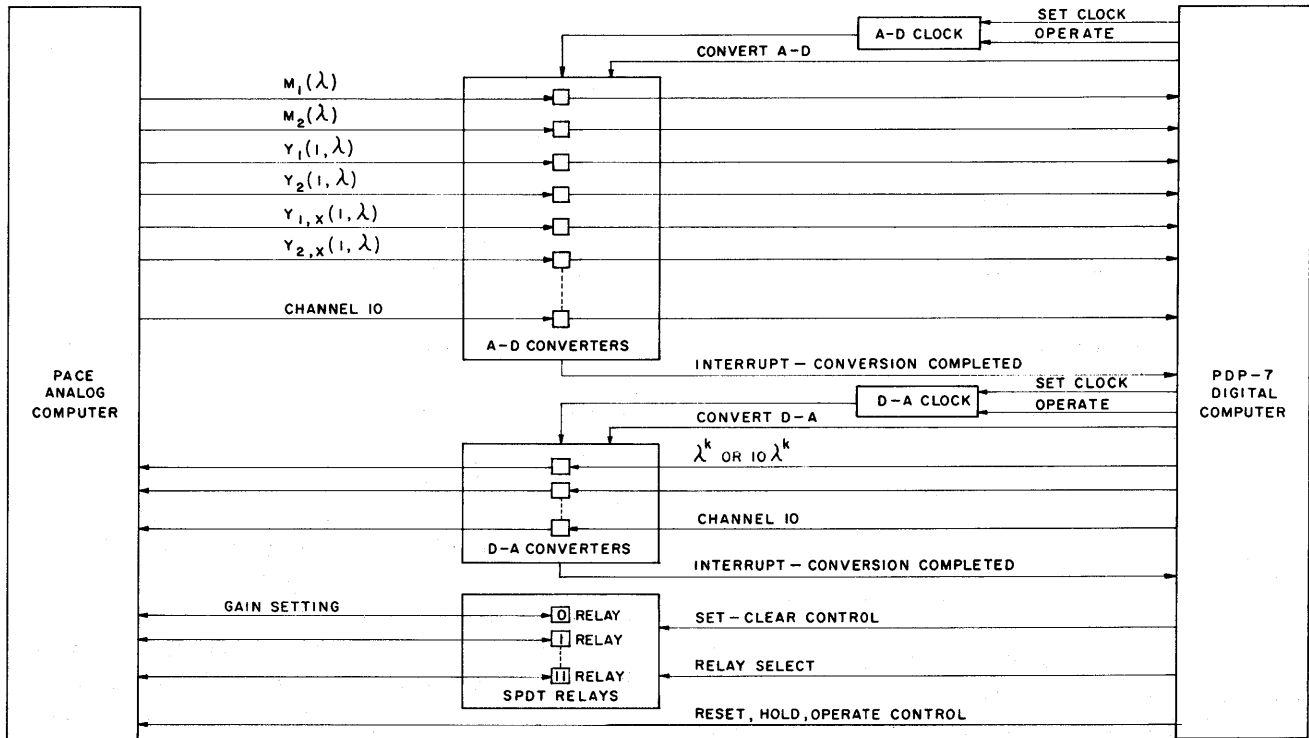


Figure 2. Flow diagram of the hybrid system.

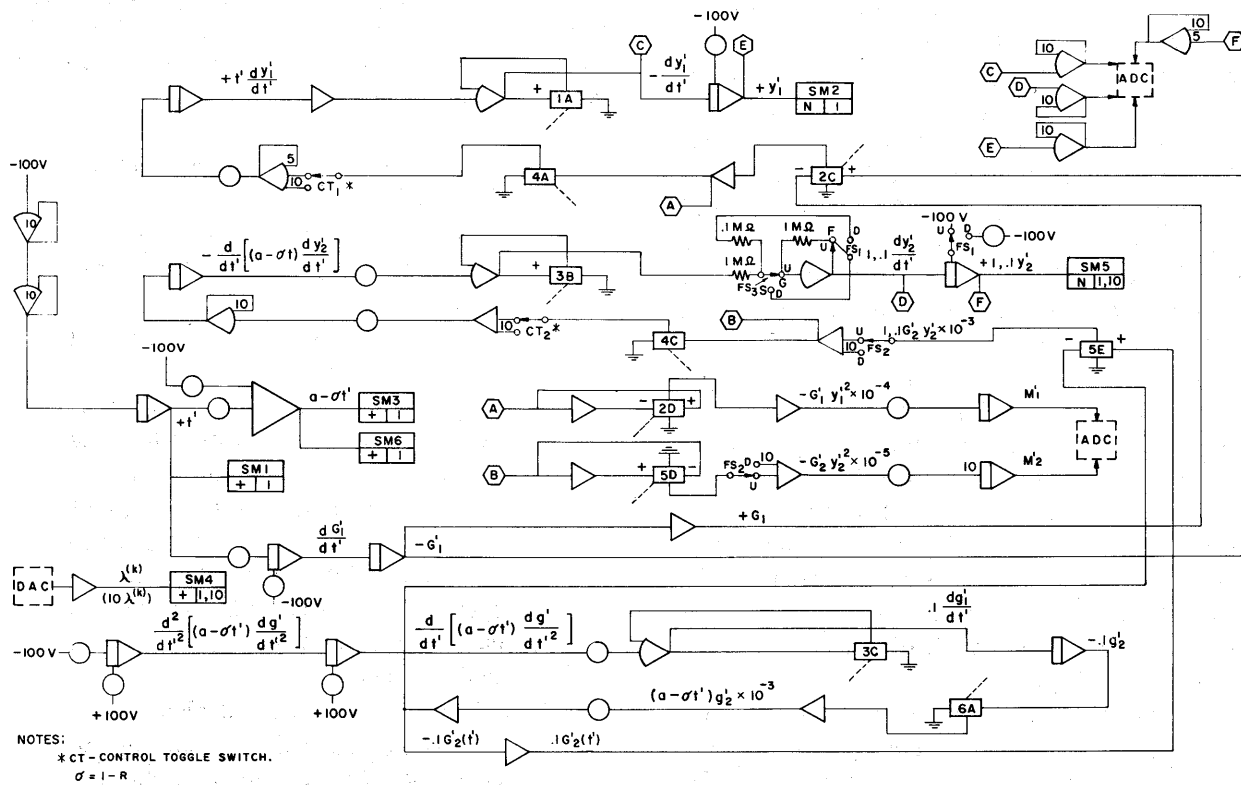


Figure 3. Analog circuit diagram.

stored program digital computer connected through a link to the analog computer.⁴ Figure 3 shows the analog computer circuit diagram including the linkage with the digital portion of the hybrid system (i.e., ADC and DAC).

The digital computer of the hybrid system is a Digital Equipment Corporation PDP-7 computer equipped with an 8K core memory. The duties of the digital computer can be summarized as follows:

1. Parameter input
2. Control
3. Storage and memory
4. Arithmetic computation
5. Output

The digital program is written in assembly language. The assembler is run on a CDC 3600 with output onto magnetic tape. The language is essentially that which is used in the D.E.C. assembler with certain mnemonics added to handle the hybrid communication. These mnemonics represent microprogrammed instructions, macros, or library subroutine calls. The arithmetic routines use single precision floating point library subroutines. The converted (A \rightarrow D) fractional numbers are "floated" before computation and "fixed" into fractional binary numbers before conversion (D \rightarrow A).

As described in the preceding section, voltages proportional to $M_i(\lambda)$, $y_i(1,\lambda)$ and $y_{i,x}(1,\lambda)$ appear as output from the analog computer. The voltage output proportional to equation variables are indicated on the analog circuit diagram (Fig. 3) as "primed" variables. The initial conditions $y_i(0)$, $y_{i,x}(0)$ and the initial values of $G_i(x)$ indicated previously are the initial conditions of the analog computer. The digital computer evaluates the eigenvalue equation $F(1,\lambda^{(k)})$ (Eq. (15)) with the $\lambda^{(k)}$ the k th approximation of λ_n and then chooses the $(k+1)$ th approximation according to the value obtained for $F(1,\lambda^{(k)})$ as described later. When $\lambda^{(k)} = \lambda_n$, the digital computer evaluates ϕ_n (Eq. (17)) and prints out values of n , λ_n^2 and ϕ_n . Thus the digital portion of the hybrid handles most of the arithmetical operations, while the analog section concentrates on solutions of the differential equations and evaluation of the integrals defining $M_i(\lambda)$ (Eqs. (18)).

In addition, the analog computer is under control of the digital computer. This is perhaps the most important single function of the hybrid system. The analog functions *reset*, *hold*, and *operate* are slaved

to those of the link connecting the analog and digital computers. Problem parameters are supplied to the analog by the digital machine via the digital-to-analog conversion equipment.

When the initial parameters have been entered and the hybrid system set up, the first iteration for λ_n is activated. The analog computer is placed in the operate mode and a clock-controlled conversion order initiated. The digital computer is in a wait loop servicing A-to-D interrupts as they occur as a result of the clock-controlled conversions. These conversions are used as a means of counting time, t' . The space variable x is related to t' by $t' = ax$, with $a = 10$ seconds. At $x = 1$ ($t' = 10.0$ seconds), the analog computer is put into "hold" and an A-to-D conversion made on the analog outputs $M_i(\lambda)$, $y_i(1,\lambda)$ and $y_{i,x}(1,\lambda)$. $F(1,\lambda^{(k)})$ is evaluated and a decision is made as to the necessity for more iterations. Hence, the problem requires not only input parameters between cases, but their modification between iterative runs. If on the basis of the value of $F(1,\lambda^{(k)})$ it is determined that no further iterations are necessary, an eigenvalue has been determined and further arithmetic calculations are made; otherwise certain input parameters are modified according to a digitally programmed algorithm and further iterations are made. The following summarizes the method used.

1. Input constants K , K_w , R , and H , which define a particular run.
2. Calculate $\gamma = RK/(1-R)$.
3. Input starting parameter λ . Reset analog.
4. Operate analog under clock control at one second intervals.
5. Put analog in "hold" at 10 intervals and calculate $F(1,\lambda)$ from the data provided by the analog computer.
6. On the basis of the value of $F(1,\lambda)$ evaluate a new λ , reset analog computer and initiate a new iteration, i.e., repeat (4) and (5) until, for properly scaled variables, $|F(1,\lambda)| < \epsilon$ for ϵ small enough.
7. Calculate ϕ_n if (6) is satisfied.
8. Increase λ_n obtained, reset analog and repeat (4)–(7) to obtain results pertaining to the next eigenvalue.

For each run results were obtained for $n = 1, 2,$ and 3 only, since these were judged sufficient for the physical problem. A general flow diagram of the digital computer program is shown in Fig. 4. Each value of n is identified as a "case" on this figure. The memory requirements are about 2K for the program.

Since the digital calculation time is negligible compared to the operate time of the analog computer (10.0 seconds) and the time allowed for the servo-mechanical multipliers to reset (2.0 seconds), the time for one iteration is approximately 12 seconds. The number of iterations per case varies from 3 to 10, so the total iteration time per case is from 0.5 to 2.0 minutes. Clearly, the most time-consuming portion of the program is the repeated searching for the roots of $F(1,\lambda)$ —i.e., for the eigenvalues. Hence the searching algorithm is of particular interest.

ALGORITHM FOR DETERMINATION OF EIGENVALUES

The computer must use some criterion for successively choosing $\lambda^{(k)}$ until, for properly scaled variables $|F(1,\lambda^{(k)})| < \epsilon$ for ϵ small enough (ϵ small determines the accuracy to which the boundary conditions at $x = 1$ are satisfied).

Several algorithms⁵ can be considered to search for the succeeding values of $\lambda^{(k)}$. These include (1) the Newton-Raphson method, (2) Regula Falsi (method of false position), and (3) an iterative trial-and-error procedure consisting of decreasing or increasing $\lambda^{(k)}$ by a fixed percentage of its previous value (10%, 1%, 0.1%, ...), determined by the value and sign of $F(1,\lambda^{(k)})$.

In most cases the Newton-Raphson method is the most efficient and converges to the root more quickly than the other methods. This method, however, requires a knowledge of the derivative of the function, and this is not available for the case being considered here. The third method mentioned converges slowly and would prove to be extremely time-consuming. Previous knowledge of the function and its behavior gave rise to the utilization of the method of Regula Falsi together with a modified iterative method similar to method 3.

The first trial is made using a starting value of λ and the eigenvalue equation $F(1,\lambda)$ is evaluated. For $0 < \lambda < \lambda_1$ it is known from previous experience that $F(1,\lambda) < 0$ and thus λ is increased or decreased by a fixed percentage, determined by the sign of $F(1,\lambda)$. Figure 5 illustrates the behavior of $F(1,\lambda)$. Succeeding runs are made in this manner until the difference in magnitude of $F(1,\lambda^{(k)})$ in two successive runs is positive and the difference in magnitude is greater than the magnitude of the most recent point; i.e., indicating that the next run may cause a sign change, or the value of $F(1,\lambda^{(k)})$ changes sign from the previous run. Regula Falsi (the method of false position) is then employed for all successive runs until convergence with the next calculation:

$$\gamma^{(k+1)} = \frac{F(1,\lambda^{(k-1)}) \lambda^{(k)} - F(1,\lambda^{(k)}) \lambda^{(k-1)}}{F(1,\lambda^{(k-1)}) - F(1,\lambda^{(k)})}$$

The reader is referred to the appropriate reference⁵ for a detailed explanation of the method of false position.

In addition to the aforementioned methods, a test is also made to determine the difference between $\lambda^{(k)}$ on successive iterations. If the difference is less

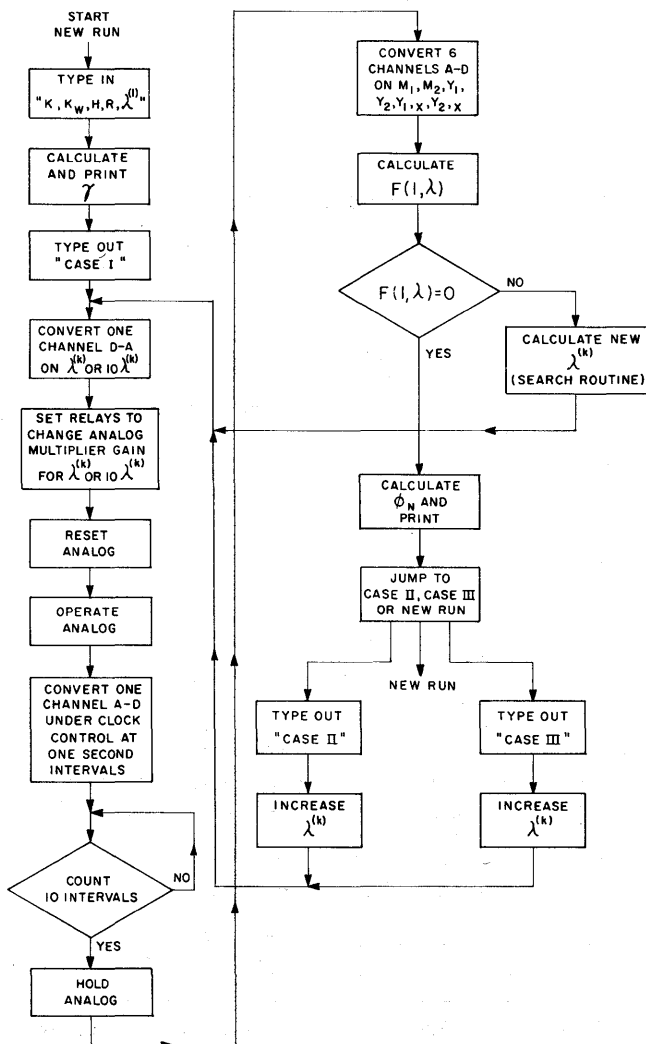


Figure 4. Flow diagram of the digital computer program.

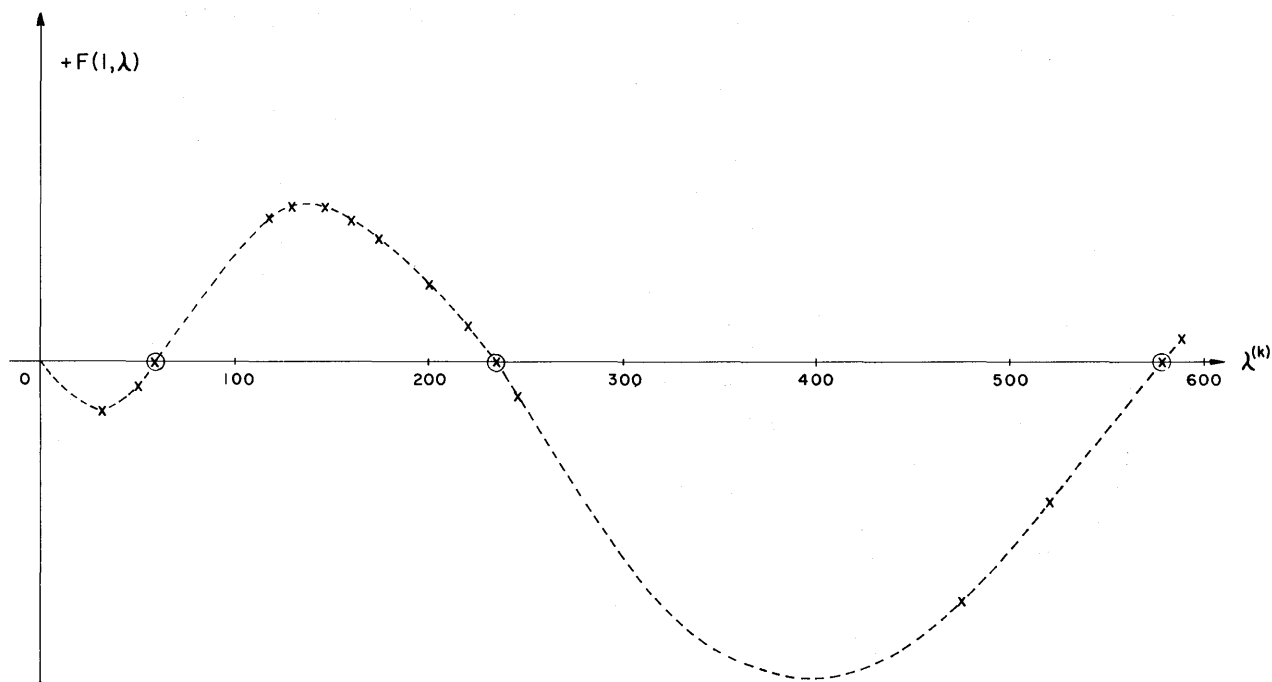


Figure 5. Behavior of $F(1, \lambda)$, ($F(1, \lambda)$ vs $\lambda^{(k)}$)

than δ , for $\delta < \epsilon$, we assume the present value of $\lambda^{(k)}$ to be the root regardless of the value of $F(1, \lambda^{(k)})$. This is particularly useful when the slope of $F(1, \lambda^{(k)})$ is large at the root and further runs will not significantly increase the accuracy of $\lambda^{(k)}$. A flow diagram of the search routine is shown in Fig. 6.

Although the digital computer performs the necessary arithmetic operations to calculate $F(1, \lambda)$, $F(x, \lambda)$ is generated on the analog computer. This allows us to illustrate the convergence of the algorithm used in the simulation. A typical $F(x, \lambda)$, for various iterations, is shown in Fig. 7. Convergence of $F(1, \lambda)$ (see Fig. 8) is illustrated by plotting the end points of $F(x, \lambda)$ vs. $\lambda^{(k)}$. $F(x, \lambda_n)$ is shown in Fig. 9 for three consecutive eigenvalues.

NUMERICAL RESULTS

Table I shows results of the system. Comparison is made with known eigenvalues as indicated in the table. The indication is that solutions obtained by the method presented here are quite adequate.

SUMMARY

It is shown that the difficulties discussed in the introduction are overcome through the utilization of a

hybrid computer. An iterative procedure is formulated on the digital elements for the trial-and-error search together with all necessary arithmetical operations. The differential equations are solved on the

Table 1. Table of Eigenvalues and Comparisons

R	K	H	λ_1	λ_2	λ_3	λ_1^*
1.0	0.1	0.1	11.37	36.05		11.27
1.0	1.0	0.1	10.99	30.47		10.94
1.0	1.0	1.0	4.129	16.54		4.134
1.0	1.0	0.5	5.822	18.40		5.81
0.90909	0.5	0.1	11.27	34.15	65.27	
0.90909	0.5	1.0	5.073	19.97	47.22	
0.6667	0.5	0.1	11.30	34.53	66.15	
0.6667	0.5	1.0	5.15	20.42	49.17	
0.6667	0.1	1.0	6.105	23.88	58.13	No
0.6667	2.5	0.1	10.34	21.79	46.71	comparison
						available
0.5	1.0	0.1	11.19	31.90	57.90	
0.5	1.0	0.5	6.16	19.63	47.75	
0.3333	2.0	0.1	10.91	27.04	50.44	
0.3333	10.0	1.0	1.109	8.228	14.26	

* R. P. Stein, "The Groetz Problem in Co-Current-Flow Double Pipe Heat Exchangers," ANL 6889 (Sept. 1964), p. 31, Table 1.

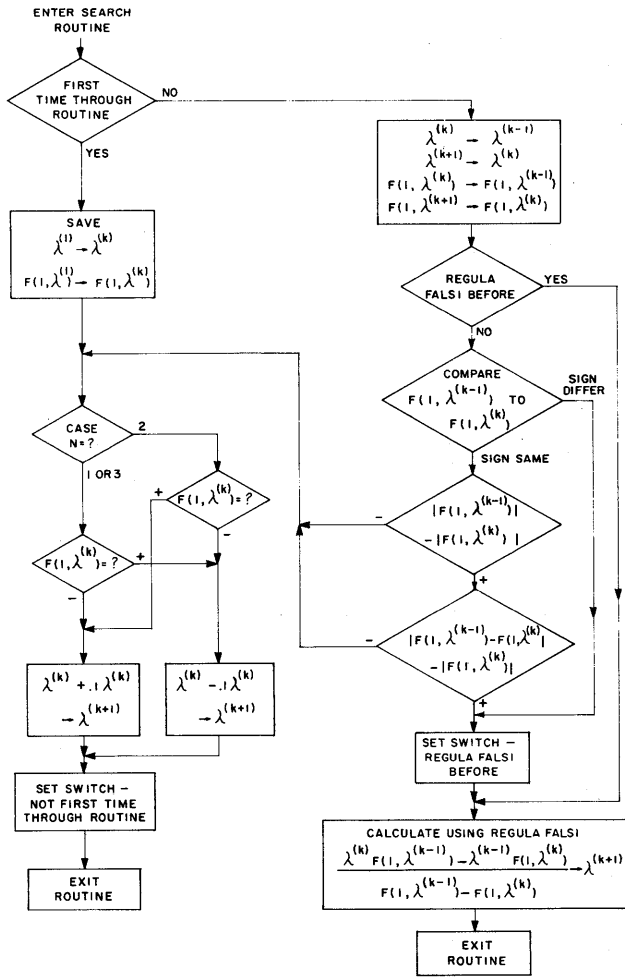


Figure 6. Flow diagram of the search routine.

analog elements of the hybrid system. Control of the system is left to the digital computer.

A measure of the success of the method is the speed at which the iterative process converges. The rate of convergence has been shown to be quite rapid even when using a "slow" analog computer (EAI 131R). Sampling rate is not a factor in this problem since we are only interested in the value of $F(1, \lambda)$ i.e., the end point of $F(x, \lambda)$.

REFERENCES

1. R. P. Stein, "The Graetz Problem in Co-Current Flow Double Pipe Heat Exchanger," *Chem. Eng. Prog. Symp. Series*, vol. 61, p. 76 (1965).
2. —, "Liquid Metal Heat Transfer," *Advances in Heat Transfer*, Vol. III (T. F. Irvine and J. P. Hartnett, eds.), Academic Press, New York, 1966, pp. 101-74.
3. —, "Mathematical and Practical Aspects of Heat Transfer in Double Pipe Heat Exchanger," *Proceedings of Third International Heat Transfer Conference*, Vol. I, A.I.Ch.E., New York, 1966, p. 139.
4. L. W. Amiot, et al, "The Argonne Hybrid Computer Maintenance Manual," Applied Mathematics Division Technical Memorandum No. 115 (Nov. 1965).
5. K. S. Kunz, *Numerical Analysis*, McGraw-Hill, New York, 1957, Chap. 1.

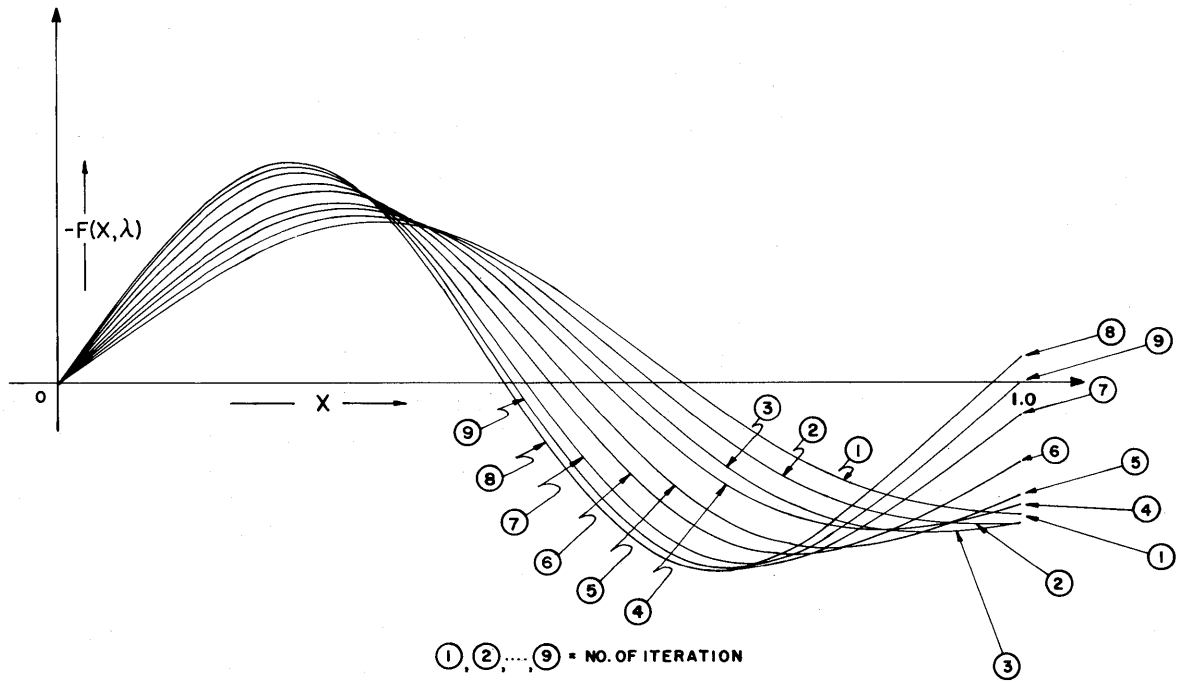


Figure 7. Convergence of $F(x, \lambda)$ for $n=2$.

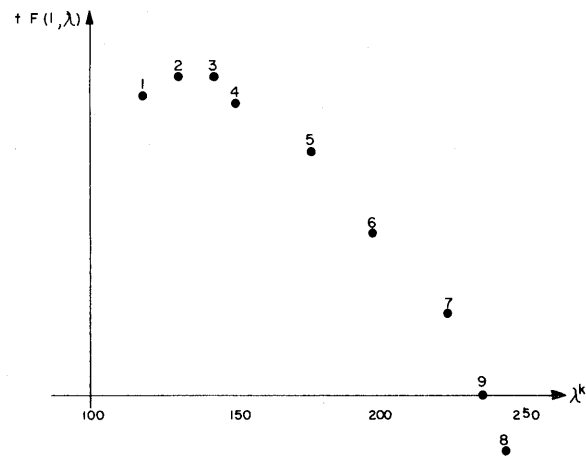


Figure 8. Convergence of $F(1, \lambda)$ for $n=2$.

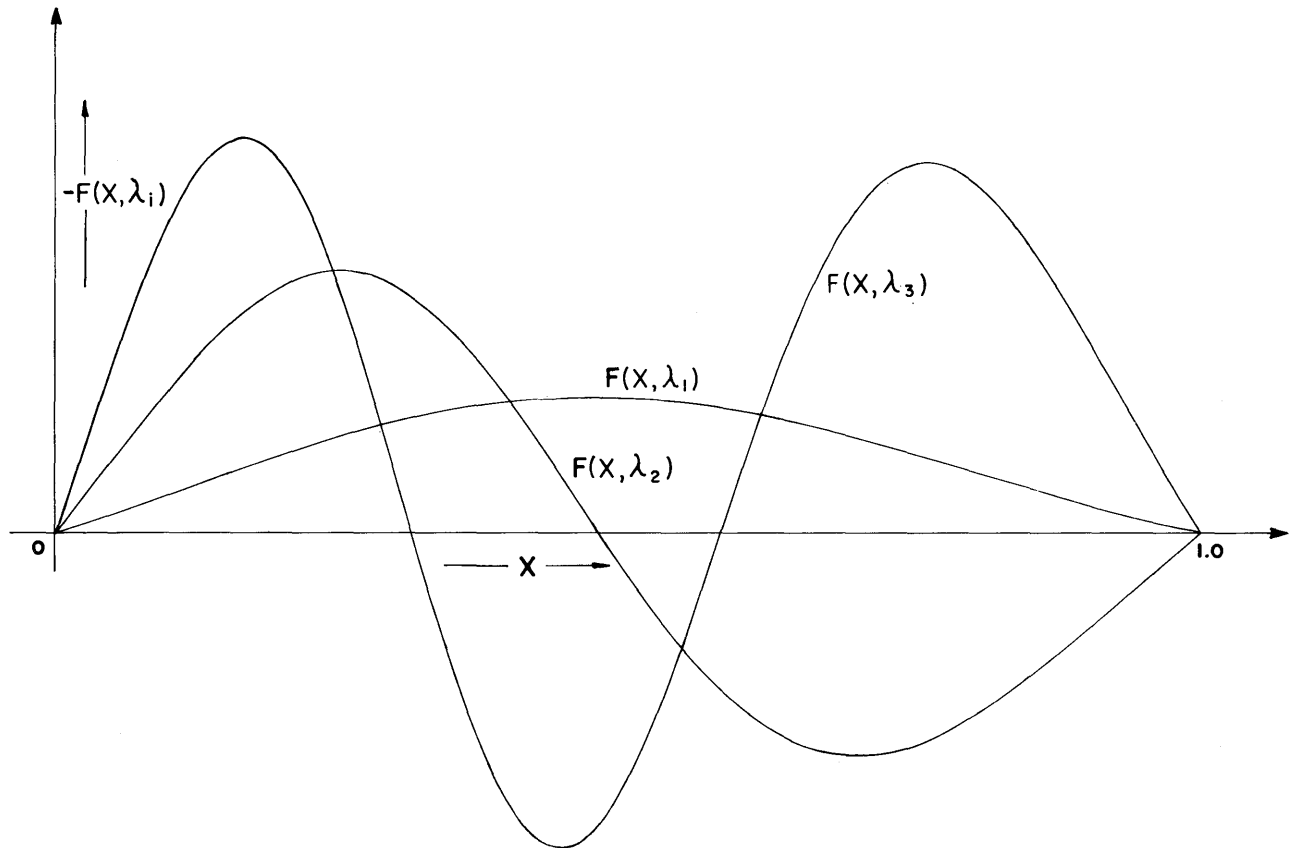


Figure 9. Computer solution of $F(x, \lambda)$, $0 \leq F(x, \lambda) \leq 1$.

A GENERAL-PURPOSE ANALOG TRANSLATIONAL TRAJECTORY PROGRAM FOR ORBITING AND REENTRY VEHICLES

Arthur I. Rubin and Lloyd Shepps

*Martin Company
Baltimore, Maryland*

INTRODUCTION

Analog computer programs for very complex flight simulations are well known. As a recent example, the complete six-degree-of-freedom equations of motion have been developed by Fogarty and Howe,¹ as well as simple two-dimensional, or two-degree-of-freedom, simulation equations. The former are useful for simulation analyses with a man in the loop, generally in real time. The latter are useful for student analyses of trajectories. The engineering trajectory analyst, however, often appears to be interested additionally in intermediate complexity. He is interested in what we shall call a pseudo-six-degree-of-freedom trajectory program, wherein the three translational degrees of freedom are handled exactly, with all terms included, but the rotational equations of motion are eliminated. In their stead, the analyst inserts arbitrary functions for three angles such as alpha (angle of attack), beta (angle of sideslip), and sigma (roll angle about the velocity with respect to air vector). Such an analytical program turns out to be of great interest in the design phase of an aerospace vehicle, since it permits evaluation of many hardware tradeoffs among different configurations. Such a program, in existence at the Martin Company in digital form since 1964, is described by Wagner and Garner.²

The development of an equivalent analog program was recently undertaken, not as an exercise to show that an analog computer can do what a digital computer can do, but rather to give the analyst a more efficient tool. One advantage sought was a reduction in time of several orders of magnitude required to obtain a complete footprint for a particular vehicle configuration and assumed reentry conditions. A second advantage sought, which would follow almost automatically if the above advantage were obtained, was to achieve a significant cost reduction in doing the job on the analog computer rather than the digital computer. The constraints that were put upon the analog program by the analyst were that it must be able to handle exactly the same inputs as the digital program, cover the same range of the variables as the digital program, and produce at least the same output as the digital program.

In order to meet these objectives, an analysis of the analog program approach as developed by Fogarty and Howe was made. It was decided that the existing six-degree-of-freedom program was more complicated than was needed by the engineering analyst. Consequently it would be more expensive to use, and probably more cumbersome. On the other hand, the simplified version of the program as derived by Fogarty and Howe was too simplified for

the analyst. It was therefore necessary to start with the translational degrees of freedom trajectory equations as derived by Fogarty and Howe, and rederive the generation of the forces acting on the vehicle, allowing the analyst to impose arbitrary angles of attack and sideslip and an arbitrary roll angle about the velocity vector with respect to the air. The second addition to the program, as required by the analyst, was to compute, in addition to the normal output, which was available from the basic equations of motion, the "downrange" and "crossrange" variables as traced on the earth.

As fallout from the development of this analog program, we can make direct comparisons, because of the existence of an equivalent digital program, with digital runs to compare (1) accuracies obtained, (2) time saved, and (3) costs for obtaining solutions by each of the two distinctly separate methods. In the past, the lack of equivalent analog and digital programs for a particular problem has been one of the drawbacks in obtaining effective comparisons.

THE EQUATIONS OF MOTION

H-Frame and Earth-Frame Axes

The equations of motion are integrated in the H-frame coordinate system as discussed by Fogarty and Howe³ and are the version that they have developed.

The H-frame is a vehicle-centered coordinate system whose x -axis is horizontal and points in the direction of the horizontal component U_h of inertial vehicle velocity. The z -axis points to the center of the earth and gives the direction of the vertical component W_h of inertial vehicle velocity. The U_h, W_h plane therefore contains the inertial velocity vector V_I . The U_h component of velocity makes an angle ψ_h with respect to true north. These relationships are shown in Fig. 1. The y -axis of the H-frame is horizontal and perpendicular to the U_h, W_h plane. Since it contains no velocity component, it is not shown on the figure.

A right-handed local earth frame can be defined which has north as its x -axis, east as its y -axis and down as its z -axis. This axis system, together with the components of V_I projected on the local earth frame, namely U_e, V_e and W_e , is also shown in Fig. 1.

The H-frame is used for the trajectory computations because it has a number of advantages. These have been discussed by Fogarty and Howe³ and

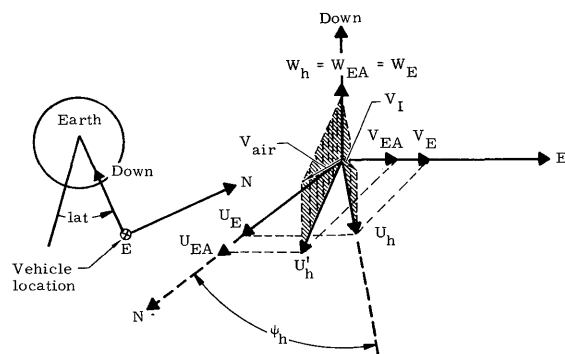


Figure 1. Right-handed local earth frame (N, E, down; V_I = inertial velocity with component U_h in horizontal plane, W_h in vertical plane; U_h makes angle ψ_h with north. E-frame components of V_I are shown as U_E, V_E, W_E . Velocity with respect to the air mass V_{air} , also shown, with its E-frame components U_{EA}, V_{EA}, W_{EA} .

are repeated here for completeness. The first advantage is due to the use of the angular momentum integral as an input to the U_h computation. For out-of-atmosphere studies, this eliminates one integration from the computation and, therefore, reduces the principal source of drift for the in-plane motion. The second advantage lies in the ability to write the equations of motion in terms of the deviations of the variables from their values in a reference orbit, thereby greatly improving the scaling. These two advantages taken together greatly improve the computational accuracy.

A third advantage, which is true for any coordinate system that points toward the center of the earth and is therefore true for the H-frame, is due to the fact that the main gravity force term can be inserted directly into the integration that produces the downward component of velocity, thereby bypassing any errors that may result from force transformations. Furthermore, by taking advantage of the perturbation form in which the equations of motion are written, the gravity term is analytically subtracted from the centrifugal force term for the reference circular orbit, thereby producing an analog computer program which is extremely precise for out-of-atmosphere and/or near-orbit conditions.

Direction Cosines, Force Transformations, Downrange and Crossrange Calculations

The derivations of these equations and/or functions are shown in Appendix 1. Nomenclature and definitions of symbols are given in Appendix 2.

CAPABILITIES OF THE PROGRAM

The following problems are examples of those that can be set up to be solved with the analog program:

1. Two-body orbital mechanics
2. Reentry from orbit
3. Boost phase
4. Synergetic maneuvers

The orbital problems (no atmosphere) can be run on one computer console (at half the cost). Orbit transfers can be accomplished by fitting the end conditions as viewed on an oscilloscope. Orbit variation due to "insertion errors" is readily calculated. The flow diagram for orbital studies only is indicated within the dotted line on Fig. 2.

The reentry phase requires two computer consoles and can be used to analyze a wide variety of problems. Reentry heating data may be obtained using the modification of Chapman's expression for convective stagnation heating. Necessary reentry locations for desired landing sites are very easily obtained by trial and error processes, optimization of reentry maneu-

vers can be performed, and control laws can be added for reentry trajectory improvements.

Future usage may require a third computer console to include the necessary equations for boost phase and synergetic maneuvers. The additions to the information flow diagram necessary for synergetic maneuvers are shown in Fig. 3.

Accuracy

To better understand the quality of the analog reentry and orbit simulation, a comparison can be made with the existing digital program, which uses the IBM 7094 Model II (a high-speed digital). Though differing forms of the equations are used, their mathematical equivalence has been verified by a comparison of output data from the two programs, a sample of which is shown in Fig. 4. The important initial oscillations are identical. This is significant since it determines that the heating rates, dynamic pressure, and total accelerations are in agreement when they are most critical. After 3 1/2 cycles, deviations appear, with the final times for touchdown differing by 1.5%. Experiment has shown that 1.0%

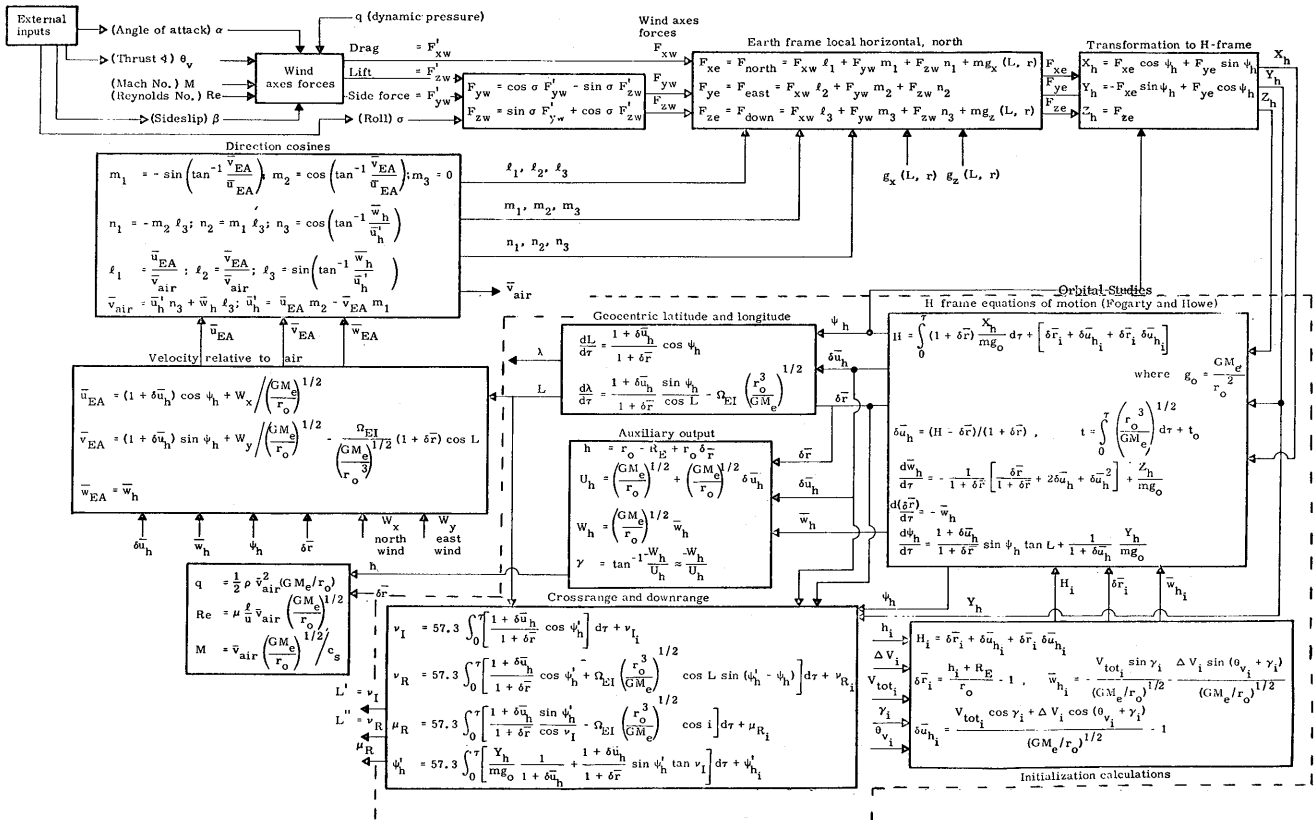


Figure 2. Information flow diagram for pseudo-six-degree-of-freedom analog trajectory program.

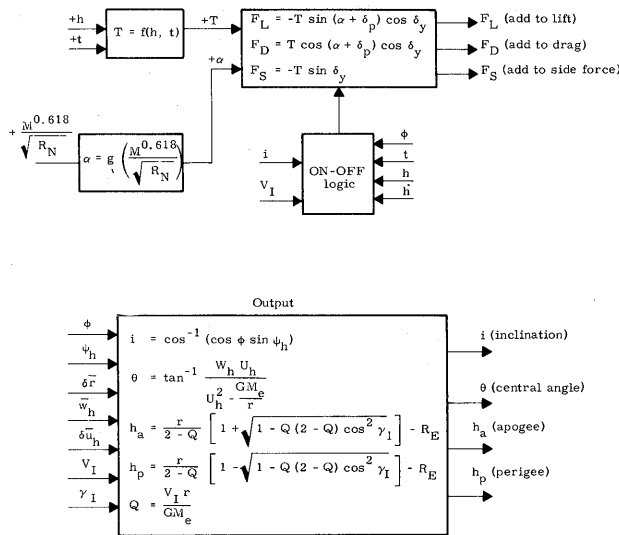


Figure 3. Flow diagram additions for synergetic maneuvers.

alterations to the aerodynamic coefficients will result in changes to the trajectory which are significantly greater than 1.5%. Since the accuracy is within the known tolerance on the system inputs, we have a useful engineering tool.

Solution Time

If any one feature makes an impression on the engineering analyst, it is the high-speed solution time of the analog. With the digital program, solution time will depend on the complexity of the run. With the analog, the complexity of a particular type of run may require more equipment but solution time will remain unchanged. An average analog solution time will be approximately 0.3 second using high-speed repetitive operation. Minimum reentry solution time for digital is approximately 6 minutes, and in special cases may be as high as 18 minutes.

When running in repetitive operation, the program reinitializes immediately after a solution and restarts automatically. This allows a turnaround time of less than 100 milliseconds with changes to the program being automated to alter the solution. Graphical solution, such as the altitude versus velocity diagram, can be displayed on the oscilloscope, giving the analyst instant feedback. This has advantages over a listing of output in numerical form, since anomalous behavior is rapidly detected, and the cause thereof can be immediately investigated.

Setup Time

We have been assuming above that the analog program is in the computer, has been checked out, and is running. If this is not the case, an hour of checkout time must be allotted. This has been accomplished effectively in double-shift operation with a manual pot system. What has reduced checkout time has been the use of card-programmed DFG's, which assured repeatability of the aerocoefficients, which are the most sensitive inputs. If the program has not been running over an extended period, such as one month, and is then returned to the computers, the setup time becomes longer—possibly as long as 8 hours. This is the time necessary for the programmer to familiarize himself with all the input/output controls.

COST

Economics is of prime importance for the often used orbital and reentry simulation. We find that an average reentry solution costs from \$0.05 to \$1.00 (depending on the time scale used) on the analog, and a minimum of \$50 on the digital.

To the trajectory analyst, the comparison is often made in terms of cost per footprint. The analog can be set up to give these footprints directly (with no off-line plotting or data reduction necessary). A footprint can be defined by approximately 100 reentry runs. Analog time per 100-run footprint is less than 5 minutes. Digital time is 10 hours. This results in an analog cost advantage of approximately 1000/1. For example, the analyst can define a landing zone for various maneuvers at a cost of about \$5.00 on analog, instead of \$5000 on digital. However, because of the extreme times and costs for a digital footprint, our analysts judiciously select five

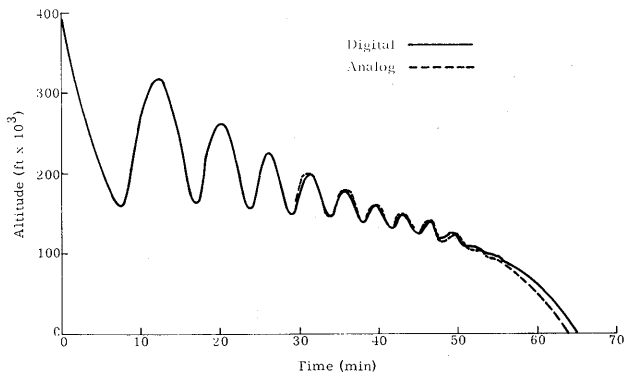


Figure 4. Analog/digital comparison of altitude versus time for a typical reentry from 400,000 feet.

runs to define a digital footprint. This requires special "fairing in" of the complete footprint curves by hand (when obtained digitally), but does reduce the digital cost to \$250 per footprint. In this instance, analog can only claim a 50/1 cost advantage, with a subtle added advantage, since the engineer need not spend the time "fairing in" the analog footprint.

INPUT AND OUTPUT DATA

The rapid solution time of the analog requires that the output information be extracted efficiently (in a matter of milliseconds) so that computation time is not lost in obtaining readout. This may be accomplished by using track-store devices to trap maximum values, end conditions, and other significant information which may then be plotted while the runs are being made.

It has been important to print out maximum values and final values for use in design considerations. The circuitry for these readouts is shown in Figs. 5a and 5b. In Fig. 5a, the first track store will follow the problem variable until the computer resets, at which time it will hold its value (the final value) until the second track store can pick it up. The second will then hold until the next reset period, at which time it will again trap the new final value. The output of track store No. 2 can now be plotted continuously and will be the change in the end condition resulting from automatically altering the program's input. Maximum values can be obtained as in Fig. 5b. The input variable q will be followed until

the output has become greater than q . At this time the large integration rate is removed and the maximum value of q is held. This maximum value can then act as an input to the track store circuits described above to obtain the maximum value for a particular run. Continuous plots of these final and maximum values can be made while operating the computer repetitively.

Numerous examples of this type of output are contained in Figs. 6 through 9. Among these figures it can be noted that for the out-of-atmosphere orbiting case, the entry flight path angle is plotted in Fig. 6 as a function of the necessary deorbit thrust angle for 28 different thrust magnitudes. The value of γ is trapped when h equals 400,000 feet. This complete graph would take approximately 50 hours of digital running time (0.02 hours for one point on one curve). By "sweeping" thrusting angle, the analog can complete one of these traces in approximately 2 minutes and the complete plot (for 28 values of ΔV) in about 1 hour.

Similar information can be obtained for the re-entry cases. Here rectilinear earth maps may be used as the plotting surface (as in Fig. 7). Examples are shown of final latitude versus longitude (Fig. 7), time for reentering (Fig. 8), and maximum heating rate (Fig. 9). Each curve represents 320 individual reentry solutions with maximum roll angle being "swept" from $+80^\circ$ to -80° (see "Sweep Circuitry" below for a further discussion of this). This is useful to indicate the degree of maneuverability for various vehicles. The five separate curves are for different lift-to-drag ratios.

Time-history recorders have been automated so that they will record two successive runs after 10% change on the input (in this case, roll angle). This allows additional detailed information as to the time shape of the trajectories.

Efficient usage of the high-speed analog requires that inputs be changed automatically. For example, to locate a landing site in reentering, one might start with 0° latitude, 0° longitude, and reenter. The landing site, from this initial condition, may turn out to be far from the desired location. The iterations required to land at the selected site are both costly and time consuming to do digitally. But on the analog they are accomplished in a matter of minutes, even when making the input changes by hand, because of the instant feedback to the analyst which is achieved

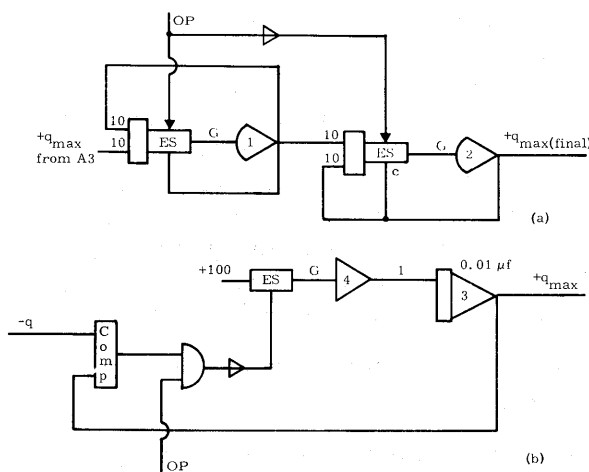


Figure 5. Printout or plot-out circuitry for footprints.

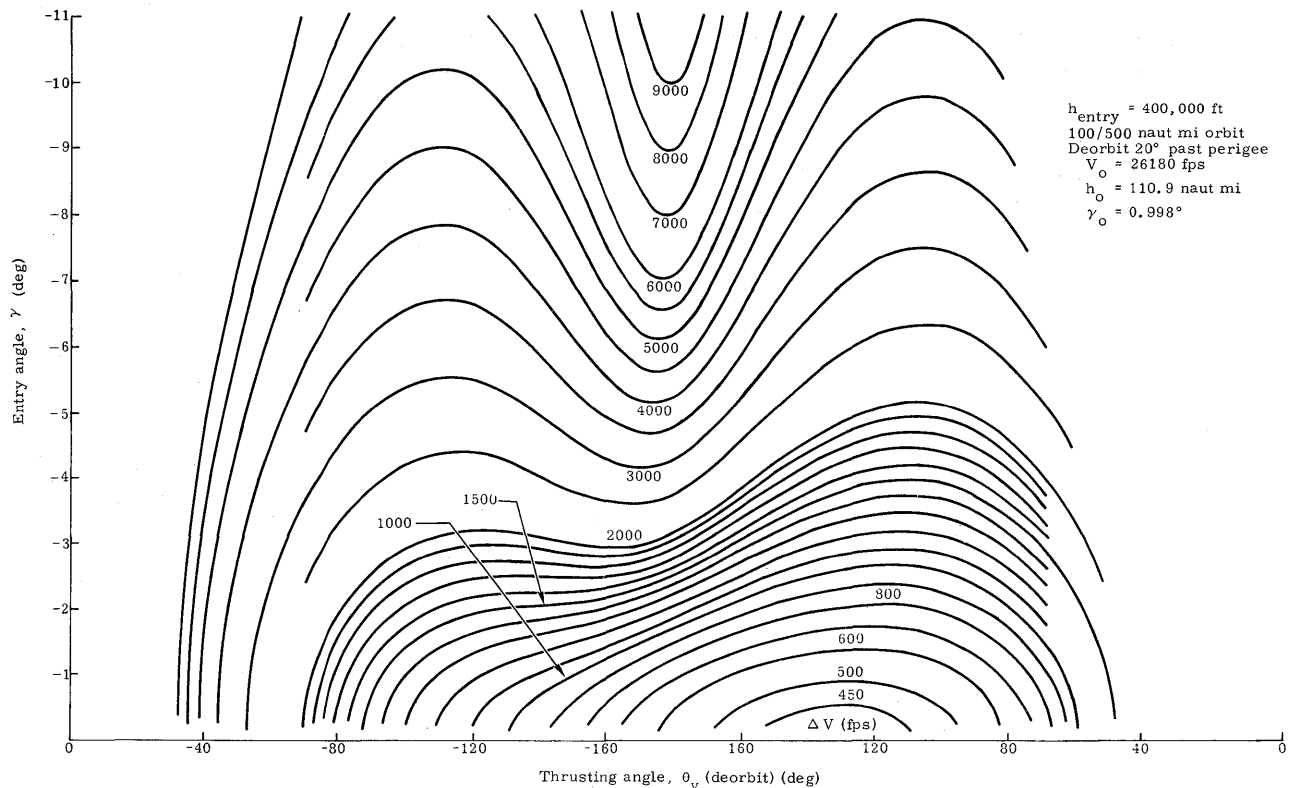


Figure 6. Entry angle γ versus thrusting angle θ_v for various values of deorbit thrust producing the velocity change ΔV . ΔV is the parameter of this figure.

by the use of high-speed repetitive/iterative analog computation.

Sweep Circuitry for Changing Roll Angle

To elaborate on what is meant by a "sweep" of roll angle, Fig. 10 will be useful. With this arrangement, Integrator 1 will be integrating at controllable rate (Pot 1) whenever Switch 1 is positioned for a roll angle sweep. The output of Integrator 2 will track Integrator 1 during the problem reset cycle and will hold during the operate cycle. Integrator 2 will be operating in synchronism with the rest of the problem integrators in the repetitive operation mode. Its output will resemble the time history shown in Fig. 11. The output of Integrator 2 (Fig. 10) is then multiplied by $\sigma_{P(\max)}$, which is the programmed roll angle (generally taken as a function of velocity) which produces the maximum cross-range. The output σ is then a variable during each solution but differs from $\sigma_{P(\max)}$ by the constant percent value stored in Integrator 2. If the percent value is zero, the output roll angle σ will be identically zero. If the percent value is +100%, then σ will be identically equal to $\sigma_{P(\max)}$.

PROBLEM EQUIPMENT ASSIGNMENT

The fact that the problem can be conveniently divided between the two computer consoles is of economical interest. The out-of-atmosphere equations of motion with the downrange and crossrange calculations and the necessary output circuits can be incorporated into one 180-amplifier computer (80 uncommitted amplifiers), while the second console can be set up to contain the direction cosines, the wind axis transformations, and the aerodynamic force calculations. This allows the out-of-atmosphere problems to be run with one computer. Large volumes of parametric handbook data have already been generated using only this orbital part of the program. A third computer might be used for synergetic maneuvers, more complicated control laws, more sophisticated aerodynamics or any combination of these.

COMPUTER SETUP FOR MAXIMUM ACCURACY

By setting up the analog and comparing it with digital results, it was discovered that particular care must be taken in the generation of the lift, drag,

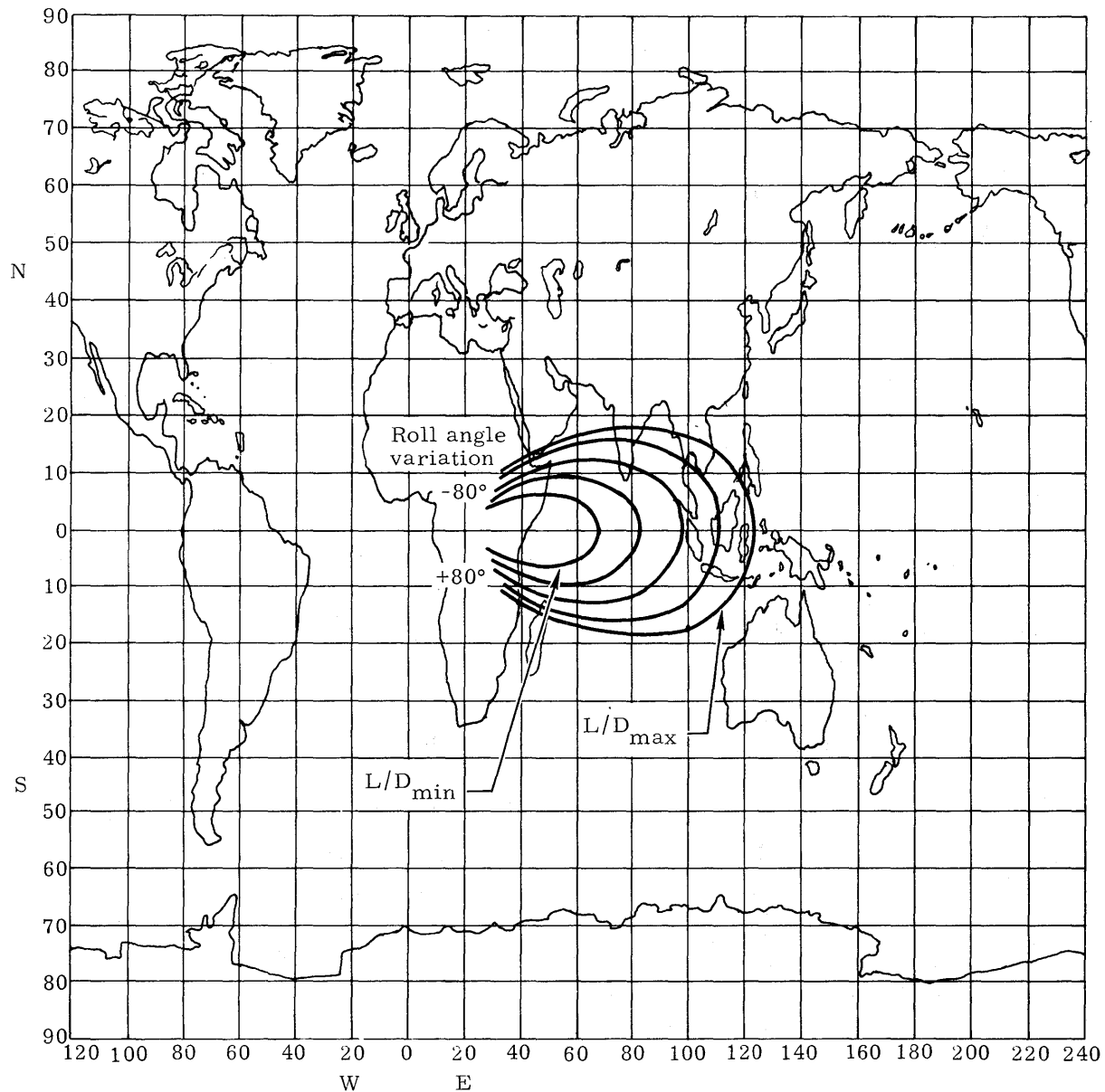


Figure 7. Typical footprint plotted automatically on a rectangular map of the earth. Footprints shown are for different values of L/D . Each footprint represents a fixed value for L/D , and the roll angle ϕ is allowed to vary from $+80^\circ$ to -80° .

side forces, and the atmosphere representation. It was also found necessary to keep the problem variables (particularly the accelerations) well scaled for a wide variety of runs.

The aerodynamic force and dynamic pressure are both calculated, using logarithms to obtain additional accuracy. The following equations define these variables:

$$F = qAC_d$$

$$q = 1/2\rho V^2$$

When ρ is large, V is small, and vice versa. When q is large, C_d is small, and vice versa. If these expressions are handled with multipliers, one of the inputs will always be small and the output will not be well scaled. By taking the log of both sides of these expressions and summing, we can normalize q and F so the scaling will be optimized. The normalized computer circuit for generating the lift and drag forces is shown in Fig. 12. Note that neither C_L nor C_D is generated explicitly, but the functional depend-

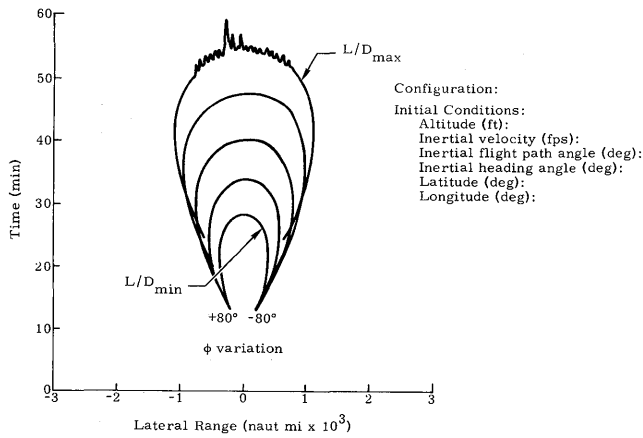


Figure 8. Time for reentry plotted as a function of lateral range for various values of L/D (each curve obtained allowing ϕ to vary from -80° to $+80^\circ$).

ence of the log of C_D and the log of C_L on Mach number (or angle of attack) are.

The accuracy of the solution was extremely sensitive to the scaling on the accelerations. By normalizing the forces and dynamic pressure, one can keep the accelerations well scaled for either short-duration or long-duration reentry trajectories.

Inverse Resolver Circuits

Another improvement to accuracy in our analog solution for the trajectory problem was obtained by the use of inverse resolution circuits, instead of squaring, square root, and, in some instances, division circuits for the generation of the direction cosines. An example of such a circuit for m_1 , m_2 and \bar{U}'_h is shown in Fig. 13. For further discussion of the direction cosines, see Appendix 1.

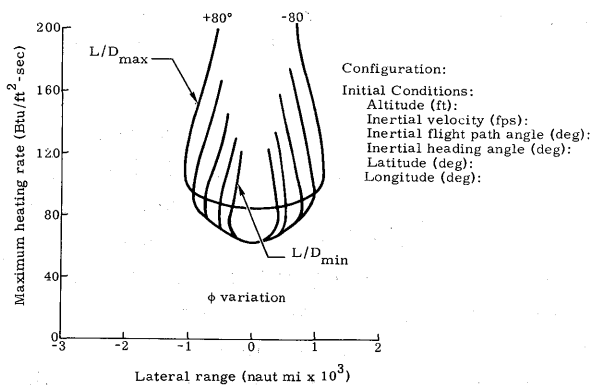


Figure 9. Maximum heating rate versus lateral range for various values of L/D (each curve obtained allowing ϕ to vary from $+80^\circ$ to -80°).

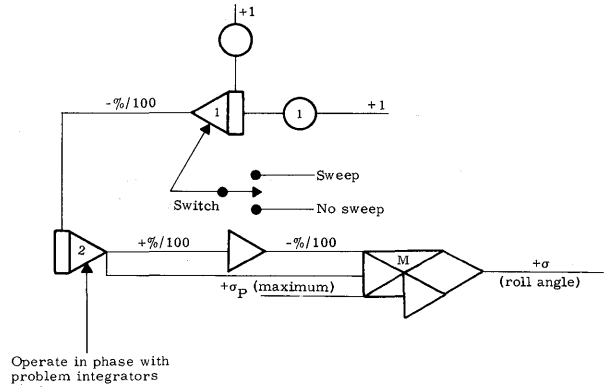


Figure 10. Sweep circuitry for automating change to roll angle.

Continuous Resolver Circuit

Since maneuvering is permitted, the heading angle Ψ_h is not constrained to lie within $\pm 180^\circ$. Since Ψ_h only feeds back into the information flow via its sine and cosine, a continuous resolution circuit was used as in Fig. 14. Here, the input to the circuit is the true $\dot{\Psi}_h$, as computed by the motion equations, which is used as is or inverted under control of the flip-flop (FF) to produce $-\dot{\Psi}_{hCR}$, where the subscript CR stands for continuous resolution. This output, when integrated, produces Ψ_{hCR} , which is the input angle to the sine and cosine generators for the continuous resolution. Note the sign of the input rate to the integrator is reversed when the output of the integrator reaches 180° (90 volts) and that the sign of $\sin \Psi_h$ is simultaneously inverted at this switching point. High-speed electronic gates, comparators, and one digital flip-flop make this circuit practical for high-speed repetitive operation.

Scaling for orbital runs depends upon the orbit's eccentricity. For reentry running, which ordinarily begins at an altitude of 400,000 feet, a 200,000-foot circular orbit is used as a reference.

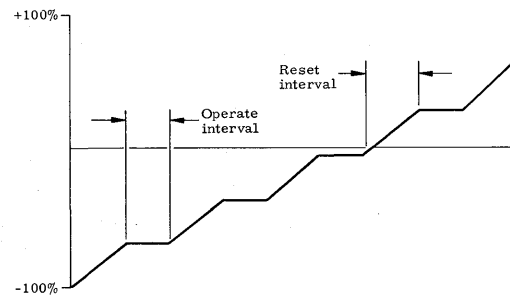


Figure 11. Sweep integrator output time history.

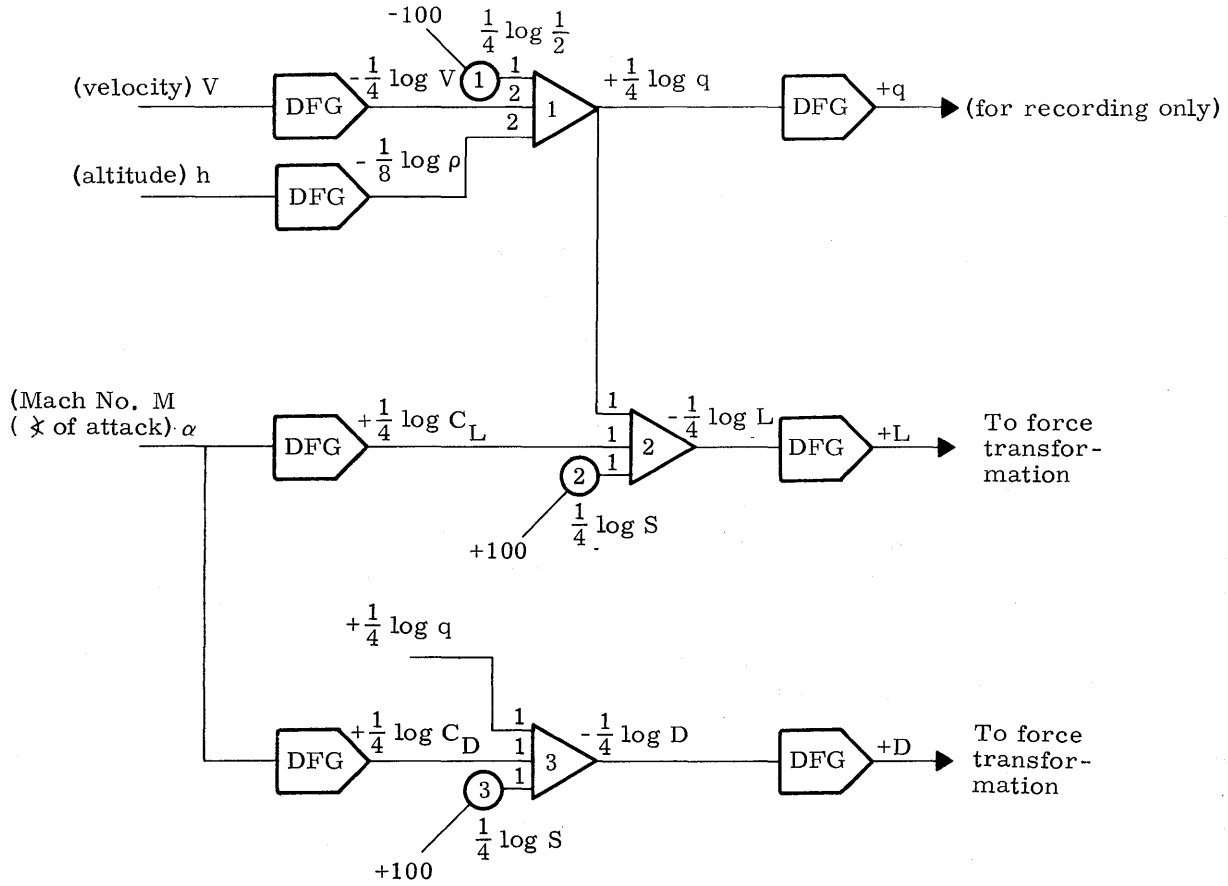


Figure 12. Force and dynamic pressure generation using logarithms.

APPENDIX 1

DERIVATION OF EQUATIONS USED FOR PSEUDO-SIX-DEGREE-OF-FREEDOM TRANSLATIONAL TRAJECTORY PROGRAMS

The earth frame components of velocity are defined from the H-frame variables U_h , Ψ_h and W_h in Eq. (1).

$$\begin{aligned} U_E &= U_h \cos \Psi_h \\ V_E &= U_h \sin \Psi_h \\ W_E &= W_h \end{aligned} \quad (1)$$

Velocity with Respect to Air

For a near-earth satellite moving through the earth's atmosphere, the velocity with respect to the air is in general not equal to the inertial velocity, either in magnitude or direction, because of the fact that the earth's air mass is moving with the earth's rotation, with a velocity which is dependent upon the latitude as given by Eq. (2).

$$W_r = -\Omega_{EI} r \cos L \quad (2)$$

where r is the distance from the center of the earth to the vehicle, L is the latitude, and Ω_{EI} is the rotational rate of the earth. The minus sign indicates that the observer in the inertial frame thinks he is moving west with respect to the air mass.

If winds can be superimposed, we can define a wind from the north as a positive wind W_x , and a wind from the east as a positive wind W_y . The components of velocity with respect to the air, U_{EA} , V_{EA} , W_{EA} , are shown in Eq. (3).

$$\begin{aligned} U_{EA} &= U_E + W_x \\ V_{EA} &= V_E - \Omega_{IB} r \cos L + W_y \\ W_{EA} &= W_E = W_h \end{aligned} \quad (3)$$

These are the three components of the velocity with respect to the air which is shown as V_{air} in Fig. 1. The components of the vehicle velocity with respect to the air vector projected on the earth frame are U_{EA} , V_{EA} and W_{EA} . It is assumed at present that there is no W_z wind (either an updraft or a downdraft).

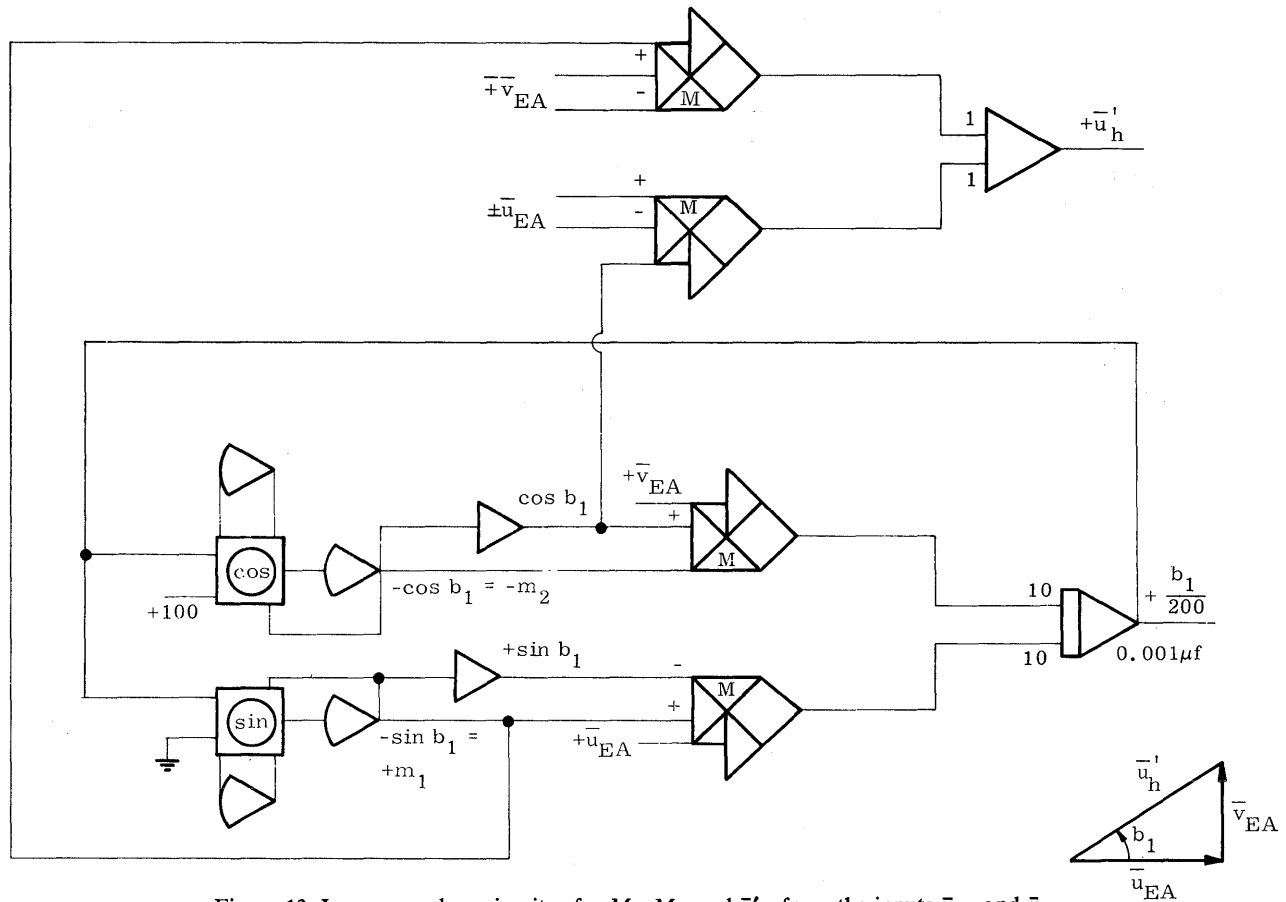


Figure 13. Inverse resolver circuitry for M_1 , M_2 , and \bar{u}'_h , from the inputs \bar{u}_{EA} and \bar{v}_{EA} .

Wind Axes and Aerodynamic Forces

Let us now define a wind axis coordinate system which has its x -axis vector in the direction of V_{air} . Rather than complicating Fig. 1 by drawing the wind axes superimposed in Fig. 1, we draw the wind axes separately in Fig. 15 and relate them to the vector V_{air} in the figure, which shows the wind axes for both the banked and the unbanked cases. The i_w and k_w vectors form a vertical plane, but neither i_w nor k_w need be horizontal or vertical. Vector j_w , since it is perpendicular to a vertical plane (unbanked vehicle), must be parallel to the local horizontal plane. Sigma is the bank angle from the local vertical plane, which can be arbitrarily specified by the analyst. In general, therefore, the analyst, having specified alpha, beta and the bank angle sigma, has then specified the wind axis forces F_{xw} (drag), F_{zw} (lift) and F_{yw} (side force). These three forces will appear on the i_w axis, the k_w axis and the j_w axis, respectively, with proper signs to take into account opposition to the motion. For example, drag is opposite to V_{air} , and lift is generally opposite to k_w . This

is because k_w positive is in a downward direction. The banked wind axis forces F'_{yw} and F'_{zw} must be resolved through the roll angle sigma into the true wind axis frame i_w, j_w, k_w . The transformation equations are given in Eq. (4).

$$\begin{aligned}
 F_{xw} &= F'_{xw} \\
 F_{yw} &= \cos \sigma F'_{yw} - \sin \sigma F'_{zw} \\
 F_{zw} &= \sin \sigma F'_{yw} + \cos \sigma F'_{zw}
 \end{aligned}
 \tag{4}$$

Direction Cosines for Force Resolution

Now all that remains is to resolve the true wind axis forces F_{xw}, F_{yw}, F_{zw} into the earth-frame coordinate system. We must therefore derive the direction cosines, which relate the true wind axis coordinate system (namely, the i_w, j_w, k_w vector system) to the E-frame system (the earth-frame system), that system which has north, east and down as its three mutually perpendicular directions. If we define i_n, j_e and k_d as the unit vectors along the earth-frame coordinate axes, we then have by definition the vector i_w as shown in Eq. (5).

$$\vec{i}_w = l_1 \vec{i}_n + l_2 \vec{j}_e + l_3 \vec{k}_d
 \tag{5}$$

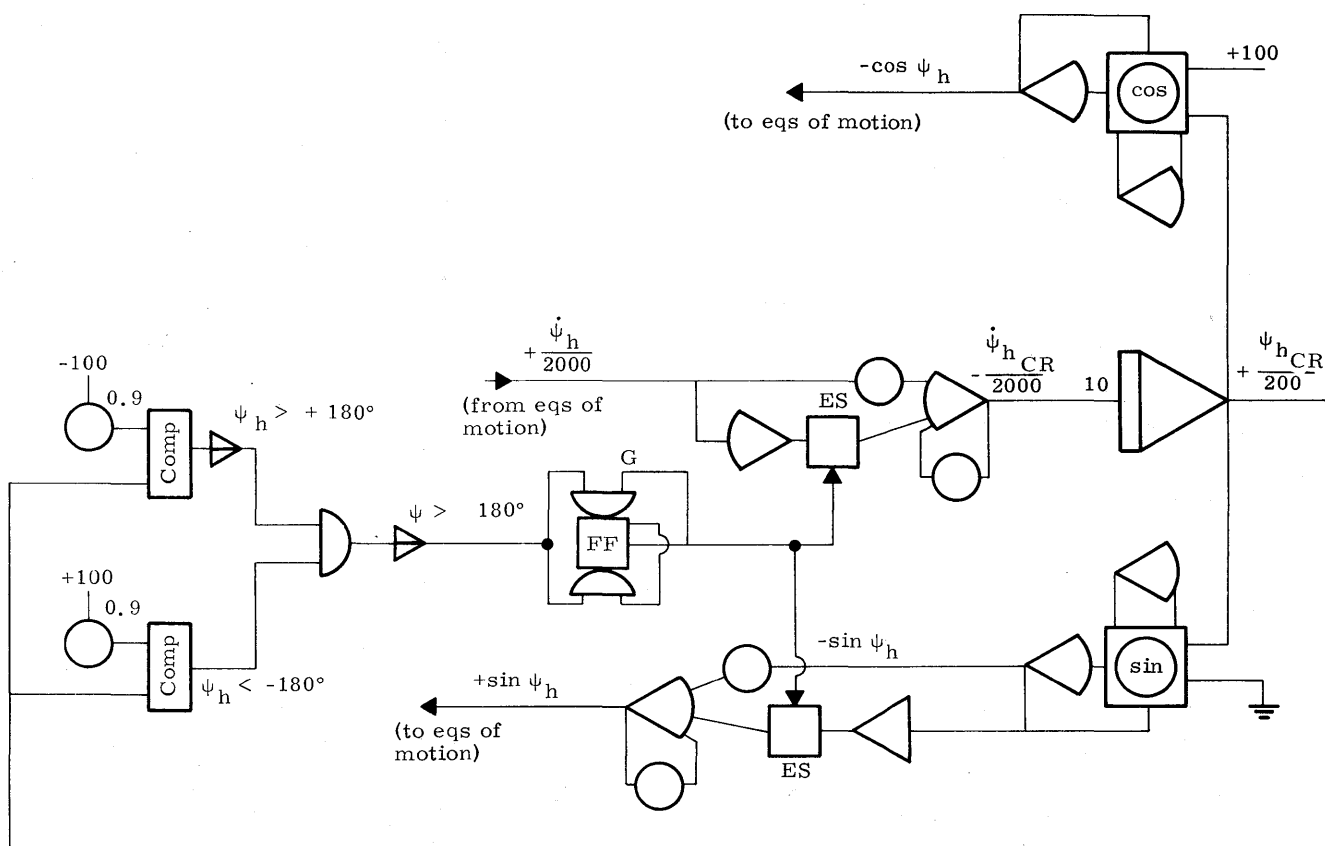


Figure 14. Continuous resolution circuit.

The direction cosines l_1, l_2 and l_3 define the direction of V_{air} with respect to the E frame. By inspection of Fig. 1, it is obvious that l_1, l_2 and l_3 are given by the relations in Eq. (6).

$$\begin{aligned} l_1 &= U_{EA}/V_{air} \\ l_2 &= V_{EA}/V_{air} \\ l_3 &= W_{EA}/V_{air} = W_h/V_{air} \end{aligned} \quad (6)$$

Furthermore, the magnitude of V_{air} is given by Eq. (7).

$$V_{air} = (U_{EA}^2 + V_{EA}^2 + W_h^2)^{1/2} \quad (7)$$

The remaining direction cosines are obtained by resorting to geometry. By definition, we know that n_3 is the projection of k_w upon k_d . These two vectors are contained in the vertical plane, which also contains the velocity vector relative to the air, V_{air} . This vertical plane is shown in Fig. 16, from which it can be seen that the projection of k_w upon k_d is given by the relation shown in Eq. (8).

$$n_3 = \cos a_1 = \frac{U'_h}{V_{air}} \quad (8)$$

In order to obtain the direction cosines n_1 and n_2 , which are projections of k_w upon i_n and j_e , it is necessary to first find the projection of k_w on the horizontal plane. This is equal to the projection of k_w upon U'_h . Upon inspection of Fig. 16, we find the projection of k_w upon U'_h is equal to $-\sin a_1$ which is equal to $\frac{-W_h}{V_{air}}$.

Now let us draw the horizontal plane which contains U'_h, i_n and j_e . The horizontal plane is shown in Fig. 17. By inspection of the figure, we can now project any vector along U'_h , on both i_n and V_{EA} . In particular, if we have the projection of k_w upon U'_h and then further project upon i_n , we then have n_1 . If we project the component of k_w upon U'_h to j_e , we have n_2 . By trigonometry, then, the relations (9) and (10) define the direction cosines n_1 and n_2 .

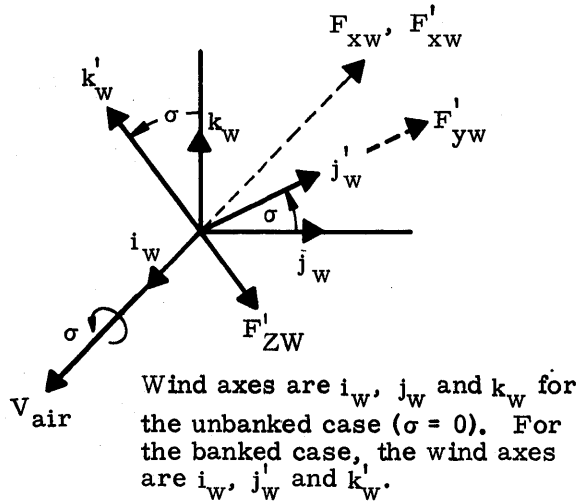


Figure 15. Wind axes.

$$n_1 = \left(\frac{-W_h}{V_{air}} \right) \cos b_1 = \left(\frac{-W_h}{V_{air}} \right) \left(\frac{U_{EA}}{U'_h} \right) \quad (9)$$

$$n_2 = \left(\frac{-W_h}{V_{air}} \right) \sin b_1 = \left(\frac{-W_h}{V_{air}} \right) \left(\frac{V_{EA}}{U'_h} \right) \quad (10)$$

Since j_w is parallel to the horizontal plane and perpendicular to U'_h , by inspection of Fig. 17, we can then find m_1 , the projection of j_w upon i_n , and m_2 , the projection of j_w upon j_e . Note that m_3 is identically equal to zero by definition of the coordinate system i_w, j_w, k_w (Eq. (13)). By inspection of Fig. 17, then, and from the definitions of m_1 and m_2 we can write the expressions for m_1 and m_2 which are shown in Eqs. (11) and (12).

$$m_1 = -\sin b_1 = \frac{-V_{EA}}{U'_h} \quad (11)$$

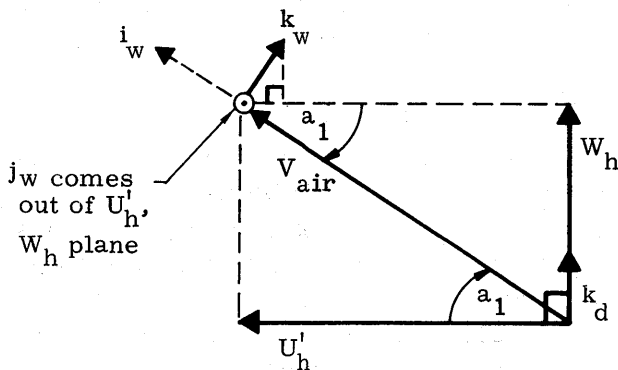


Figure 16. Vertical plane containing V_{air}, W_h , and U'_h (the projection of V_{air} on the local horizontal plane).

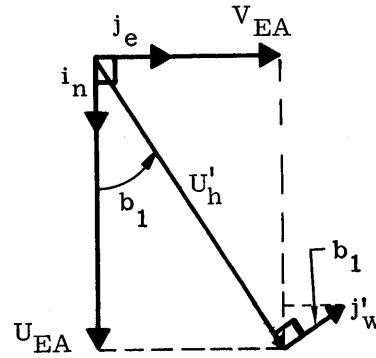


Figure 17. Vectors in local horizontal plane.

$$m_2 = \cos b_1 = \frac{U_{EA}}{U'_h} \quad (12)$$

$$m_3 \equiv 0 \quad (13)$$

Analog Computer Formulation of Direction Cosines

Because of accuracy considerations, it is undesirable to take the square root of the sum of the squares. Consequently, equivalent trigonometric forms for the direction cosines will be used. We found the most accurate form of trigonometric conversions used the following equivalent forms for U'_h and V_{air} . These are shown in Eqs. (14) and (15).

$$U'_h = m_2 U_{EA} - m_1 V_{EA} \quad (14)$$

$$V_{air} = l_3 W_h + n_3 U'_h \quad (15)$$

Here m_1 and m_2 were obtained from an inverse resolver using the relations shown in Eqs. (16) and (17).

$$m_1 = -\sin \left(\tan^{-1} \frac{V_{EA}}{U_{EA}} \right) \quad (16)$$

$$m_2 = \cos \left(\tan^{-1} \frac{V_{EA}}{U_{EA}} \right) \quad (17)$$

In a similar manner, and for similar accuracy considerations when performing these calculations via analog computers, n_3 and l_3 were calculated using an inverse resolver, with relations shown in Eqs. (18) and (19).

$$l_3 = \sin \left(\tan^{-1} \frac{W_h}{U'_h} \right) \quad (18)$$

$$n_3 = \cos \left(\tan^{-1} \frac{W_h}{U'_h} \right) \quad (19)$$

It should also be observed that n_1 and n_2 are related to the direction cosines m_2, l_3 and m_1, l_3 by expressions shown in Eqs. (20) and (21).

$$n_1 = -m_2 l_3 \quad (20)$$

$$n_2 = m_1 l_3 \quad (21)$$

The two remaining direction cosines l_1 and l_2 used the original expressions defined in Eq. (6), which are repeated here as Eqs. (22) and (23) for completeness.

$$l_1 = \frac{U_{EA}}{V_{air}} \quad (22)$$

$$l_2 = \frac{V_{EA}}{V_{air}} \quad (23)$$

Thus, Eqs. (13) through (23) define the auxiliary relations and direction cosines which allow us to project a vector from the i_w, j_w and k_w wind axis system to the local horizontal earth-frame coordinate system i_e, j_e and k_e .

Earth Oblateness Terms and Transformation to the H-Frame

The contributions to the forces acting in a northerly and downward direction and due to earth oblateness can then be incorporated into the analysis by adding the two oblateness terms to the force transformations from F_{xw}, F_{yw} and F_{zw} to F_n, F_e and F_d , as shown in the block diagram, Fig. 2. There is one final transformation of forces from the earth-frame local horizontal coordinate system to the H-frame coordinate system, as shown in the block diagram. Thus, having derived the forces acting in the H-frame coordinate system, it is only necessary to insert these forces into the H-frame equations of motion. These equations are shown in the block diagram of the entire computation (Fig. 2) under the heading "H-Frame Equations of Motion" (Fogarty & Howe³).

The Energy Integral

We had attempted to use the energy integral as a correction term to the H-frame equations of motion but, as Fogarty and Howe have found for reentering vehicles, the amount of this correction is so small as to be unnoticeable in computation. We verified that for the scaling used for reentry vehicles, the energy integral does in fact add nothing to the

accuracy of the computation. Consequently, that particular aspect of Fogarty and Howe's equations of motion is not included here. The variables as shown in the block diagram with bars over them represent normalized variables, using Fogarty and Howe's normalization factor. A complete definition of all symbols and variables used is shown in Appendix 2.

Initialization and Auxiliary Outputs

Of interest in this set of equations is the method of initialization. The block showing the initialization calculations requires as inputs all the input initial conditions shown to the right of that block. These are input conditions that the engineering analyst desires to control and/or to study. Consequently, it proved best to set up the initialization calculations directly on the analog computer to allow the analyst to insert the initial parameters that he has under his control.

To complete the derivations of the equations used for the translational study, it is then necessary to integrate the outputs of the H-frame equations of motion, namely, $\psi_h, \delta \bar{u}_h$ and $\delta \bar{r}$ to obtain geocentric latitude and longitude and, equally important, to obtain crossrange and downrange as traced on the earth. Auxiliary outputs are calculated algebraically, such as the altitude, total velocity in the horizontal plane, total vertical velocity, the flight path angle γ , the aerodynamic pressure q , the Reynolds number Re and Mach number M which, of course, are necessary for the generation of the aerodynamic forces.

Downrange and Crossrange Equations

The downrange and crossrange equations have not been discussed previously. To calculate the parameters, we must define a new local horizontal coordinate system n' and e' which we may call pseudo-north and pseudo-east. Pseudo-north axis, n' , points in the direction of the pole of the original, undisturbed vehicle orbit plane. Pseudo-east axis, e' , will then lie in the "equator" of the undisturbed vehicle orbit plane. For an inclined orbit, the geometry would appear as shown in Fig. 18 at some particular time during the orbit. The relationship between n' and e' and n and e is shown in this figure. As the orbiting vehicle travels around the earth, there would be a changing relationship between n' and n , so that at another time, for example,

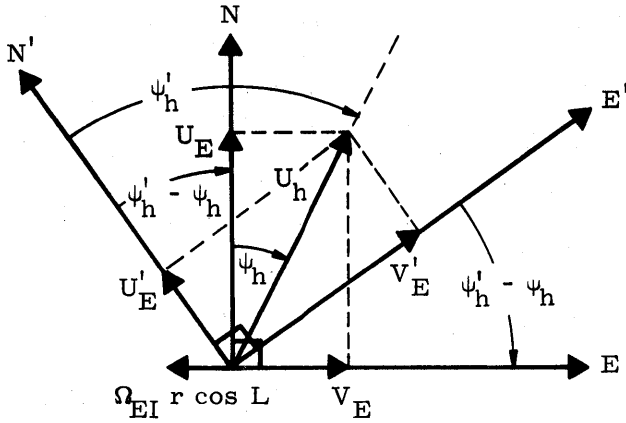


Figure 18. Horizontal plane geometry (some vehicle maneuver has been assumed so that Ψ'_h is not equal to 90°).

when the maximum latitude is achieved, which occurs when the vehicle crosses the plane containing n' and n , the geometry as seen by an observer in the vehicle would be as shown in Fig. 19.

Heading angle ψ_h will vary greatly for an inclined orbit; Ψ'_h (pseudo-heading angle) will vary only slightly since the "primed" system is the "natural system" for the particular orbit. The component of inertial velocity on the horizontal plane is still U_h . It is, of course, independent of the coordinate system used. It should be noted that ψ'_h is equal to the rate of change of the H-frame heading angle, except that it is measured with respect to the pseudo-north, pseudo-east coordinate system, instead of the true north-east coordinate system. This would be the difference, therefore, between the H-frame angular velocity along k_d , which is the same for both $\dot{\psi}_h$ and $\dot{\Psi}'_h$, and the e' -frame angular velocity along k_d . Thus,

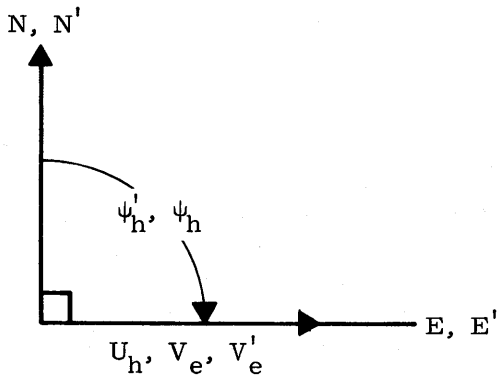


Figure 19. Horizontal plane geometry, when orbiting vehicle crosses plane containing N' and N (no vehicle maneuver assumed so that $\Psi'_h = 90^\circ$).

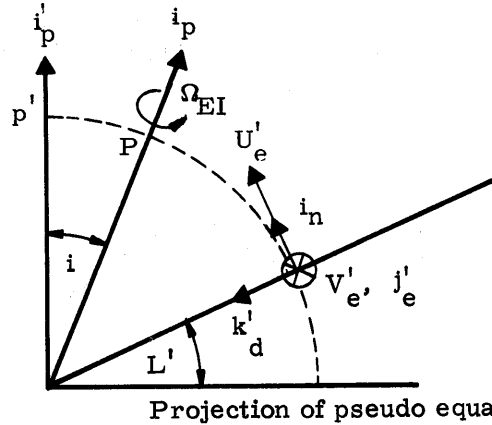


Figure 20. Vertical plane through vehicle and pseudo-pole p' .

we may write (similar to Eq. (12) of Fogarty and Howe¹) Eq. (24).

$$\dot{\Psi}'_h = r_h - \omega'_{ez} \quad (24)$$

In Eq. (24), r_h is the component of angular velocity of the H-frame in the k_d direction, while ω'_{ez} is the k_d component of the n' -, e' -frame angular velocity vector. Continuing the development of this equation as in Ref. 1, we may write the vector Eq. (25) for ω' .

$$\vec{\omega}' = \frac{V'_e}{r \cos L} \vec{i}'_{pole} - \frac{U'_e}{r} \vec{j}'_e \quad (25)$$

The geometrical relationships are shown in Fig. 20. Note that k'_d is identically the same direction as k_d (by definition). L' in the figure represents the pseudo-latitude of the orbiting vehicle in the pseudo-coordinate system. No earth rotation is included in L' since we are comparing the inertial coordinate i'_p with coordinates i'_n and k_d .

Figure 21 shows the geometrical relationships among i'_p , i'_n and k'_d and the angle L' . From Fig 21, we may write the vector Eq. (26) for i'_p .

$$\vec{i}'_p = \cos L' \vec{i}'_n - \sin L' \vec{k}'_d \quad (26)$$

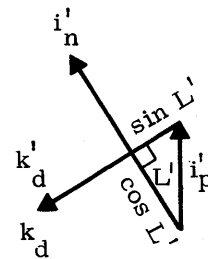


Figure 21. Detail rearrangement of Fig. 20, showing relationship of i'_p to L' , i'_n and k'_d .

Therefore Eq. (25) may be rewritten as Eq. (27).

$$\vec{\omega}' = \frac{V'_e}{r} \vec{i}'_n - \frac{U'_e}{r} \vec{j}'_e - \frac{V'_e \sin L'}{r \cos L'} \vec{k}'_d \quad (27)$$

The z component of the vector ω' is, therefore, as shown in Eq. (28).

$$\omega'_{ez} = \frac{-V'_e \sin L'}{r \cos L'} = \frac{-V'_e}{r} \tan L' \quad (28)$$

Since r_h is given in Ref. 1, Eq. (8), as Y_h/mU_h , we may write Eq. (29).

$$\frac{d\Psi'_h}{dt} = \frac{Y_h}{mU_h} + \frac{V'_e}{r} \tan L' \quad (29)$$

From Fig. 18 we may write the expressions for V'_e and U'_e shown in Eqs. (30) and (31).

$$V'_e = U_h \sin \Psi'_h \quad (30)$$

$$U'_e = U_h \cos \Psi'_h \quad (31)$$

The rate of change of L' is then given as shown in Eq. (32).

$$\frac{dL'}{dt} = \frac{U'_h \cos \Psi'_h}{r} = \frac{U'_e}{r} \quad (32)$$

Equations (29) through (32) allow us to solve for L' and Ψ'_h . The variables L' and Ψ'_h , when combined with the earth's rotation, will allow us to compute the downrange and crossrange variables as traced on the earth's surface. To obtain the downrange expressed as equivalent longitude (in radians) we need merely note that if there were no earth rotation, the expression for the downrange rate of change would be as given in Eq. (33).

$$\frac{d\lambda}{dt} = \frac{V'_e}{r \cos L'} \quad (33)$$

If the angle between i'_p (pseudo-pole) and i_p (the true earth pole) is angle i as shown in Fig. 20, then we can see that the component of the earth's rotation Ω_{EI} projected on i'_p is $\Omega_{EI} \cos i$. Thus, the rate of change of longitude, including the earth's rotation, with respect to the original orbit plane (the equivalent of downrange including earth rotation) is given by Eq. (34).

$$\frac{d\mu_R}{dt} = \frac{V'_e}{r \cos L'} - \Omega_{EI} \cos i = \frac{U_h \sin \Psi'_h}{r \cos L'} - \Omega_{EI} \cos i \quad (34)$$

The initial inclination angle i is a constant, of course, for any particular orbit chosen. The calculation of

the crossrange or equivalently the rate of change of the latitude in the pseudo-coordinate system is somewhat more complicated. We must find the velocity of an observer located on the earth's surface immediately below the vehicle. Since the earth rotates eastward, an observer on earth would observe a vehicle as traveling westward, even though its inertial velocity component in that direction were zero.

The relative velocity, with respect to an earth observer, along N' now must be derived. The component of velocity due to the earth's rotation observed by an observer on earth is along the minus east axis, and has the value $\Omega_{EI} r \cos L$. This is shown in Fig. 22. Now all that is required is to add this component of velocity opposite to V_e in Fig. 18. If we then project this component of velocity onto the n' axis (U'_e vector), we then have the total relative velocity along the i'_n vector in the pseudo-earth-coordinate system. From Fig. 18, it can easily be seen that the earth's rotational velocity $\Omega_{EI} r \cos L$ along the minus V_e direction has the component $\Omega_{EI} r \cos L \sin (\Psi'_h - \Psi_h)$ on the i'_n axis. Therefore, the crossrange as seen by an observer on the earth's surface will be given in radians by the expression shown in Eq. (35).

$$\begin{aligned} \frac{dv_R}{dt} &= \frac{U'_e + \Omega_{EI} r \cos L \sin (\Psi'_h - \Psi_h)}{r} \\ &= \frac{U_h \cos \Psi'_h}{r} + \Omega_{EI} \cos L \sin (\Psi'_h - \Psi_h) \end{aligned} \quad (35)$$

Heating Rate Equation

The stagnation point convective heating rate \dot{Q}_c as given in Ref. 4 was calculated from

$$(R_{\text{nose}})^{1/2} \dot{Q}_c = V_{\text{air}}^{3.15} (\rho/\rho_{SL})^{1/2} \frac{17,600}{V_c^{3.15}} A(\alpha) \quad (36)$$

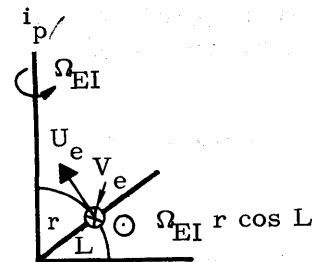


Figure 22. Vertical plane showing the direction and magnitude of velocity due to earth's rotation rate Ω_{EI} , at latitude L , as seen by an observer on earth.

where R_{nose} is the radius of the nose of the vehicle, ρ_{SL} is the density of air at sea level, V_c is an average circular orbital velocity, taken as 25,600 ft/sec, and $A(\alpha) = 1$.

The total was obtained as the integral of \dot{Q}_c . Note that in the block diagram (Fig. 2) μ_R, ν_R are shown in degrees.

Dynamic Pressure, Reynolds Number, Mach Number

The algebraic calculations for dynamic pressure q , Reynolds number Re , and Mach number M are as shown in the block diagram (Fig. 2). This completes the derivation of the equations.

NORMALIZATION OF EQUATIONS OF MOTION

Following Fogarty and Howe's practice, normalized variables are defined so as to avoid amplitude and time scale changes in going from one kind of vehicle to another. The practice is similar to that used in Ref. 1. Velocities are divided throughout by the term $\sqrt{g_0 r_0}$, which is the circular orbital velocity at the reference radial distance r_0 . The acceleration of gravity at this distance is $g_0 = GM_e/r_0^2$, where G is the universal gravity constant and M_e is the mass of the earth. The radius r is normalized by dividing by r_0 , the same reference radial distance. Time is normalized by use of the time scale factor $\sqrt{g_0/r_0}$. Computer time, T , is then related to dimensionless time, τ , by an arbitrary scale factor such as 0.01 or 0.001. This indicates that the analog computer solutions are effectively run much faster than real time. Bars over the variables signify normalization and/or nondimensionalization. In addition to normalization, following the practices of Fogarty and Howe,¹ perturbation variables are defined such as $\delta\bar{u}_h$ and $\delta\bar{r}$. The definitions of all the variables are given in Appendix 2. Equations defining the normalizations are also given there, where applicable.

APPENDIX 2

SYMBOLS

c_s Velocity of sound
 $\left. \begin{matrix} F_n \\ F_e \\ F_d \end{matrix} \right\}$ Identically equivalent to F_{xe}, F_{ye} and F_{ze} , respectively

$\left. \begin{matrix} F_{xe} \\ F_{ye} \\ F_{ze} \end{matrix} \right\}$ External forces of aerodynamic and thrust origin, plus components of gravity due to noncentral force field terms g_x and g_z , along earth-frame axes i_n, j_e and k_d
 $\left. \begin{matrix} F_{xw} \\ F_{yw} \\ F_{zw} \end{matrix} \right\}$ External forces of aerodynamic and thrust origin along true wind axes i_w, j_w and k_w . (Note: j_w is horizontal)
 $\left. \begin{matrix} F'_{xw} \\ F'_{yw} \\ F'_{zw} \end{matrix} \right\}$ External forces of aerodynamic and thrust origin along banked wind axes i'_w, j'_w and k'_w . Bank angle with respect to horizontal plane is angle σ
 G Universal gravitation constant
 g_0 Value of gravity, g , at r_0 ; $g_0 = GM_e/r_0^2$
 h Altitude of vehicle above surface of the earth
 h_a Apogee
 h_p Perigee
 h_i Initial altitude of vehicle
 H Nondimensional angular momentum
 H_i Initial value of H
 g_x, g_z Components of gravity acceleration due to noncentral force field along i_n and k_d , respectively
 i Inclination of original orbit plane with equatorial plane
 i_n, j_e, k_d Unit vectors along the three axes of the Euler (E) frame, north, east and down toward the center of earth, respectively
 i'_n, j'_e, k'_d Unit vectors along the three axes of the pseudo-Euler frame, pseudo-north, pseudo-east and down toward the center of the earth. NOTE: $k'_d = k_d$
 i_p Unit vector from center of earth through North Pole
 i'_p Unit vector from center of earth through pseudo-North Pole
 i_w, j_w, k_w Unit vectors along the three wind axes, i_w along V_{air} , j_w horizontal and k_w perpendicular to i_w and j_w
 i'_w, j'_w, k'_w Unit vectors along the banked wind axes. NOTE: $i'_w = i_w$
 L Geocentric latitude
 L' Pseudo-latitude without earth's rotation

L''	Crossrange variable (with earth's rotation)	V_{air}	Magnitude of velocity of vehicle relative to the air
$l_{1,2,3}$	Direction cosines representing the respective projections of i_w upon i_n, j_e and k_d	\bar{v}_{air}	Normalized value of V_{air} , $\bar{v}_{air} = \frac{V_{air}}{\sqrt{g_o r_o}}$
l/μ	Characteristic length/kinematic viscosity	V_c	Constant average circular orbital velocity for heating rate calculation
m	Mass of vehicle	W_h	Vertical component of inertial velocity of vehicle
M	Mach number	W_x	Horizontal wind from the north
$m_{1,2,3}$	Mass of the earth	W_y	Horizontal wind from the east
M_e	Direction cosines representing the respective projections of j_w upon i_n, j_e and k_d	X_h, Y_h, Z_h	Forces in H-frame
$n_{1,2,3}$	Direction cosines representing the respective projections of k_w upon i_n, j_e and k_d	α	Angle of attack
q	Dynamic pressure	β	Angle of sideslip
\dot{Q}_c	Stagnation point convective heat rate	γ	Inertial flight path angle
r	Distance of the vehicle from the center of the earth	γ_i	Initial value of flight path angle
\bar{r}	Normalized distance of the vehicle from the center of the earth, $\bar{r} = r/r_o = 1 + \delta\bar{r}$	$\delta\bar{r}$	Normalized perturbation of r from the nominal value r_o
r_i	Initial value of r	$\delta\bar{r}_i$	Initial value of $\delta\bar{r}$
r_o	A convenient constant reference value for r (e.g., the value of r for a nominal reference circular orbit)	$\delta\bar{u}_h$	Normalized perturbation of U_h from the nominal value $\sqrt{g_o r_o}$
Re	Reynolds number	$\delta\bar{u}_{h_i}$	Initial value of $\delta\bar{u}_h$
R_E	Radius of the earth at point at which altitude is measured	ΔV_i	The initial velocity change due to initial thrust impulse for orbit transfer or retrothrust for reentry
R_{nose}	Radius of nose of vehicle	Θ_v	Rocket thrust angle (with respect to V_{air}) in the plane of the motion
t	Real time	Θ_{v_i}	Initial value of Θ_v
T	Computer time	λ	Geocentric longitude
\bar{u}'_h	Normalized value of U'_h ; $\bar{u}'_h = U'_h/\sqrt{g_o r_o}$	λ'	Pseudo longitude without earth rotation
U_h	Horizontal component of inertial velocity of vehicle	λ''	Downrange variable
U'_h	Horizontal component of velocity of vehicle relative to air	ρ	Density of air
$V_{tot i}$	Total initial inertial velocity = $(U_{h_i}^2 + W_{h_i}^2)^{1/2}$	ρ_{SL}	Density of air at sea level
U_{EA}, V_{EA}, W_{EA}	Components of vehicle velocity with respect to the air, V_{air} , on the earth-frame axes i_n, j_e and k_d	σ	Roll angle about V_{air}
$\bar{u}_{EA}, \bar{v}_{EA}, \bar{w}_{EA}$	Normalized values of U_{EA}, V_{EA}, W_{EA} ; i.e., all velocity components are divided by $\sqrt{g_o r_o}$	$\sigma_{P(max)}$	Program roll angle to produce maximum crossrange
\bar{u}_h, \bar{w}_h }	Normalized components of velocity in H-frame; $\bar{u}_h = U_h/\sqrt{g_o r_o}$, $\bar{w}_h = W_h/\sqrt{g_o r_o}$ or $\bar{u}_h = 1 + \delta\bar{u}_h$	τ	Dimensionless time
		Ψ_h	H-frame heading angle, between i_n and U_h
		Ψ'_h	Pseudo-heading angle, between i'_n and U_h
		Ω_{E1}	Earth's rotation rate about i_p
		μ_R	Downrange variable (including earth rotation) = λ''
		v_I	Crossrange variable (inertial, without earth rotation = L')
		v_R	Crossrange variable (with earth rotation = L'')

REFERENCES

1. L. E. Fogarty and R. M. Howe, "Flight Simulation of Orbital and Re-entry Vehicles," *IRE Transactions on Electronic Computers*, vol. 11, pp. 555-63 (1962).
2. W. E. Wagner and W. R. Garner, Jr., "Near Earth Flight Analysis," Martin-Baltimore Report No. ER 12465 (June 1964).
3. L. E. Fogarty and R. M. Howe, "Space Trajectory Computation at the University of Michigan," *Simulation Journal*, vol. 6, pp. 220-26 (1966).
4. R. W. Detra, N. H. Kemp and F. R. Riddell, "Addendum to Heat Transfer to Satellite Vehicles Re-entering the Atmosphere," *Jet Propulsion*, vol. 27, pp. 1256-57 (Dec. 1957).

SATELLITE LIFETIME PROGRAM

John L. Stricker

and

Wayne W. Miessner

Martin Company, Baltimore, Maryland

INTRODUCTION

The gravitational forces of the sun, moon, and oblate earth can cause significant changes in the orbit of a near-earth satellite. These orbital perturbations can cause an early failure of the satellite. The solar and lunar perturbations vary in their effect, for any given initial orbit, according to the day and hour of injection. In fact, the change of a fraction of an hour in injection time may cause a change in the satellite's lifetime of several months. So for effective planning of a satellite mission, these perturbations must be predicted.

The classical methods of celestial mechanics,¹ in which the instantaneous position of the satellite is calculated, are not satisfactory for this problem. On digital computers, the solution is too slow and too expensive, thus limiting the amount of information that can be acquired. On analog computers, such computations carried on for long intervals of time produce excessive errors due to drift, and due to the inherent sensitivity of the solution to small errors.

The development of faster digital methods has made it possible to make many more predictions within the constraints of the time and money available. With enough information, a plot referred to as a "Launch Window" can be made. For a specific nominal launch, as injection time is changed, the

times separating the predicted successful launches from predicted failures in terms of a minimum acceptable lifetime are marked on a plot of hours Universal Time versus days of the year. By connecting these points a contour is formed separating success "areas" from failure "areas." Figure 1 shows an example of a launch window plot.

Of the various faster digital methods used above, the simplest methods calculated the mean rate of change per satellite orbit of the elements of the instantaneous osculating elliptical orbit of the satellite without calculating the satellite's instantaneous position. The launch window plot was generated by making preliminary runs covering the "area" of interest with a very coarse mesh, to determine roughly the location of the launch window boundary, and by making selected runs with progressively finer meshes until the boundary was determined with the desired precision. Because of the great number of runs required and the time required after one series of runs to select the injection times for the next series, generating a launch window plot would still require several months elapsed time.

The development of the modern iterative analog computer, with parallel digital logic, and high speed repetitive operation capability, made the automatic generation of launch windows feasible on the analog computer.

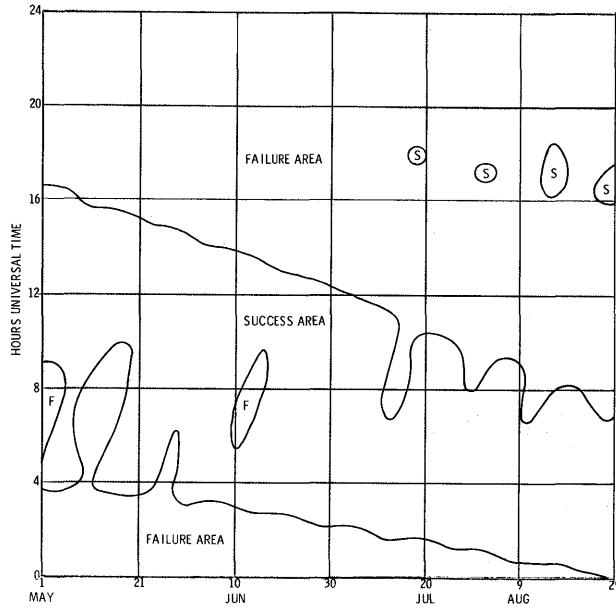


Figure 1. Launch window.

In 1964, the first such program²⁻⁴ was developed at the Martin Company by the authors, using a mathematical model of the type described above developed by M. M. Moe.⁵ The purpose of the program was to achieve a significant reduction in the cost and time required to generate the launch window plots. The program was highly successful in terms of cost, time and the number of launch windows generated. However, use of the Moe method was not completely satisfactory for analog computer use, since different reference planes were used for developing the equations for each disturbing body, requiring an excessive number of transformations, which degraded the accuracy of the calculations.

In 1965, the authors started programming the mathematical model developed by D. E. Smith.⁶ During the programming it became obvious that there were some errors in Smith's equations. The correct form of the equations was developed by C. Bowie, J. Glazer, and P. Seery of the Martin Company, and they were programmed and run by the writers as described below.

THE MATHEMATICAL MODEL

The mathematical model uses expressions for calculating the mean rate of change per orbit of the elements of an instantaneous osculating elliptical orbit. The orbital elements (see Fig. 2) are:

- a = semi-major axis
- e = eccentricity
- i = inclination to reference plane
- Ω = right ascension of the ascending node
- ω = argument of perigee.

The instantaneous rates of change of the orbital elements can be calculated from a form of Lagrange's Planetary Equations.¹ From these, equations for the mean rates of change can be developed with the following assumptions:

1. since the angular rate of the disturbing body is small compared to the angular rate of the satellite, the disturbing body may be considered fixed for one satellite orbit;
2. since the ratio of the distance of the satellite from earth center to the distance of the disturbing body is small, terms of a series expansion of the components of the disturbing body force may be truncated beyond the first few;
3. since the changes of the satellite orbits elements are small for each orbit, the values that they would have in the unperturbed orbit may be used for calculating the mean rates of change.

Letting ε represent any orbital element, the mean rate of change of the element over one satellite orbit can be expressed as

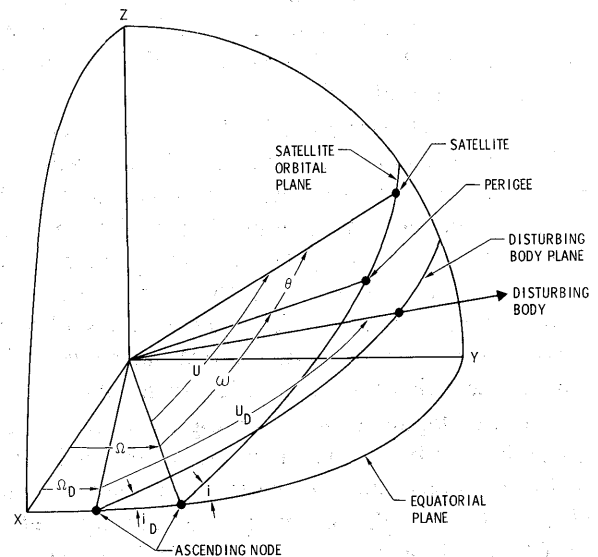


Figure 2. Satellite and disturbing body orbits.

$$\Delta \varepsilon = \frac{1}{T} \int_0^t \frac{d\varepsilon}{dt} dt = \frac{1}{T} \int_0^{2\pi} \frac{d\varepsilon}{dt} \frac{dt}{d\Theta} d\Theta$$

where Θ = true anomaly of the satellite

T = period of satellite orbit

With the above assumptions, we obtain expressions which are integrable over the true anomaly Θ .

By integrating these equations analytically, the highest frequency, that of the satellite's revolution, has been eliminated from the calculation of the satellite orbital perturbations, which makes calculation possible on the analog computer. However, it is not the frequency reduction that is significant here, but rather the elimination of the part of the equations which are highly sensitive to small errors.

According to our assumptions, these mean rates are constant for one satellite period. However, errors result from the changes of the orbital elements and the angular position of the disturbing bodies. For analog computation, with its continuous integration, we update the values of these variables continuously and instantaneously, which has the effect of reducing this error.

THE COMPUTING SYSTEM

The model was mechanized using two PACE 231R-V analog consoles. Each console contains a Memory and Logic Unit, which contains parallel digital logic elements, and enables digital control of all integrators, and highspeed repetitive operation and control. For additional parallel digital logic, one rack of Harman-Kardon Facilogic plug-in digital modules was used.

PROGRAMMING

The computing system was programmed:

1. to calculate, for a near-earth satellite, the solar, lunar and earth oblateness perturbations to the orbital elements for one year after injection;
2. to solve the equations in high-speed repetitive operation mode, using a time scale of one year equals .1 seconds computer time, and automatically changing the injection time by .2 hours before each new solution, for each day of a four month period;

3. to test continuously for failure of the satellite, where failure is defined as the perturbed perigee decreasing to where atmospheric drag becomes significant (125 nautical miles);
4. to plot automatically the times during each day of the unsuccessful launches, with hours Universal Time as the ordinate, and calendar days as the abscissa, to produce a launch window diagram;
5. to enable changing all integrator time constants simultaneously, under push-button control, to slow the solution speed by a factor of 100, for plotting time histories of the orbital elements.

Information flow in the program is shown in Fig. 3.

INITIALIZATION

The initial values of the orbital elements a , i , ω and e , were considered as constant for one launch window plot, as functions of the nominal launch trajectory. The initial value of Ω , as well as the argument of latitude of the sun and moon, in an equatorial plane based inertial reference system, need to be calculated automatically in the repetitive operation reset time. For this purpose the digital logic was used to generate three discrete variables.

T_M = count of twenty-eight day periods from the first injection day

T_D = count of days from the end of the last twenty-eight day period (reset at the end of each 28-day period)

t_D = hours Universal Time

The relative motion of the sun around the earth was assumed circular, with constant distance and angular rate. From the ephemeris:

$$\Theta_s = f(M_s) \quad (1)$$

$$M_s = k_1 + k_2 D + k_3 D^2 \quad (2)$$

$$\omega_s = k_4 + k_5 D + k_6 D^2 \quad (3)$$

$$i_s = k_7 + k_8 D \quad (4)$$

$$\Omega_s = 0.0 \quad (5)$$

$$U_s = \omega_s + \Theta_s \quad (6)$$

where

k_1, \dots, k_8 = constants

D = days from reference epoch

Θ_s = true anomaly of sun

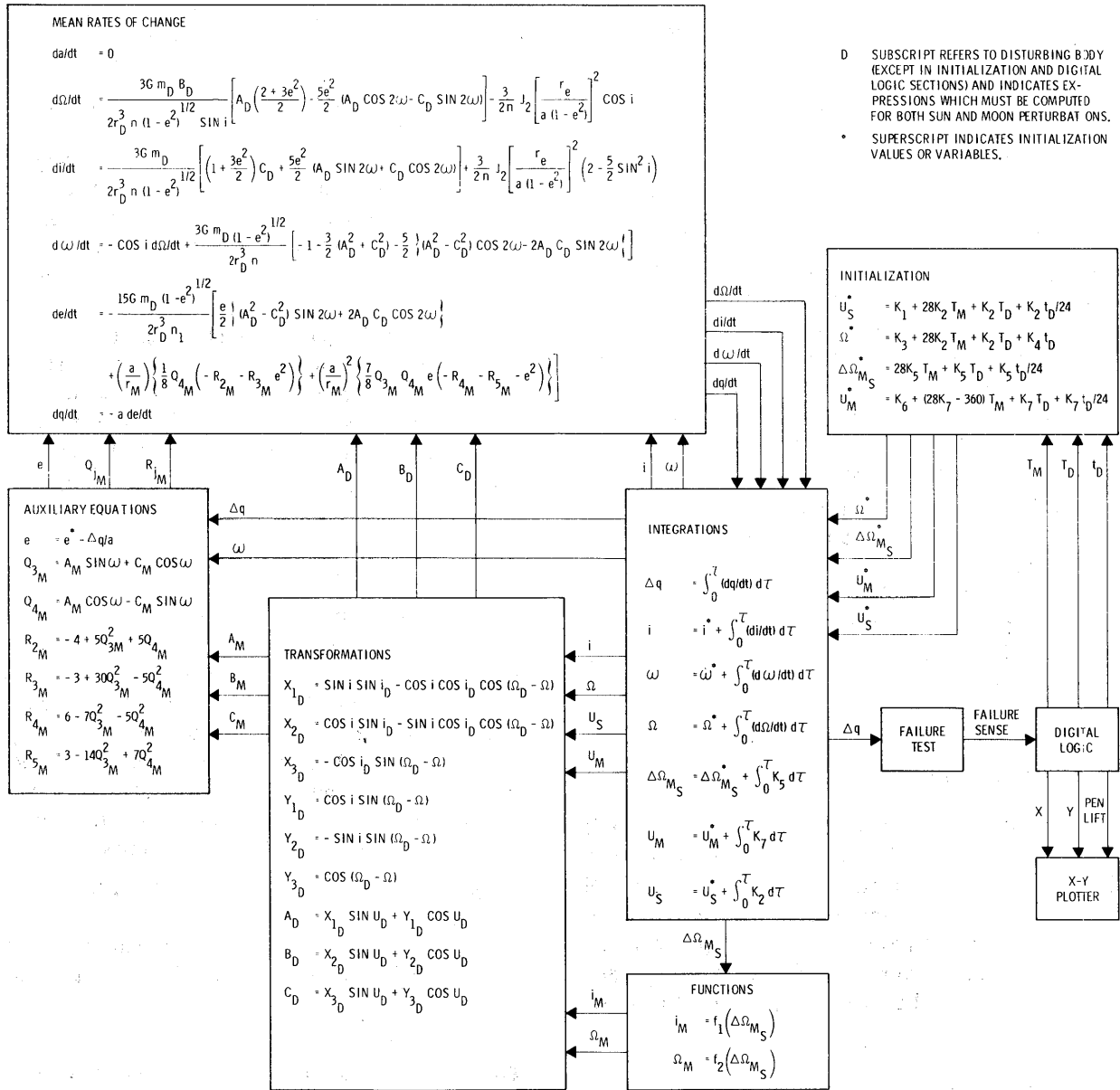


Figure 3. Information flow diagram.

- M_s = mean anomaly of sun
- ω_s = argument of perigee of sun
- i_s = inclination of the ecliptic plane
- Ω_s = ascension of right ascending node of ecliptic plane
- U_s = argument of latitude of sun

First the values were calculated for the day and hour of the first injection. Then by approximating, combining and simplifying, we obtain:

$$U_s = K_1 + 28K_2 T_M + K_2 T_D + K_2 t_D/24 \quad (7)$$

Ω_s and i_s are considered constant.

The initial value of satellite Ω is a function of the mean rates of the earth's revolution and rotation, and the nominal launch trajectory.

$$\Omega = K_3 + 28K_2 T_M + K_2 T_D + K_4 t_D \quad (8)$$

For the motion of the moon around the earth, we also assume constant distance and angular rate. From the ephemeris:

$$\odot_M = f(M_M) \quad (9)$$

$$M_M = k_9 + k_{10} D + k_{11} D^2 \quad (10)$$

$$\omega_{M_s} = k_{13} + k_{14} D + k_{15} D^2 \quad (11)$$

$$i_{M_s} = k_{16} + k_{17}D + k_{18}D^2 \quad (12)$$

$$\Omega_{M_s} = k_{19} + k_{20}D + k_{21}D^2 + k_{22}D^3 \quad (13)$$

$$U_{M_s} = \omega_{M_s} + \Theta_M \quad (14)$$

Where

- Θ_M = true anomaly of moon
- M_M = mean anomaly of moon
- ω_{M_s} = argument of perigee of moon (ecliptic plane reference)
- i_{M_s} = inclination of lunar orbit plane (ecliptic plane reference)
- Ω_{M_s} = ascension of right ascending node (ecliptic plane reference)
- U_{M_s} = argument of latitude of moon (ecliptic plane reference)

Since ω_{M_s} , i_{M_s} , Ω_{M_s} , and U_{M_s} use an ecliptic plane reference system, and we need these variables in an equatorial plane reference system, we must perform the necessary transformations. From Fig. 4, approximating i_{M_s} as a constant, spherical trigonometric relationships can be used to show that Ω_M , i_M , and U_{M_1} where

$$U_{M_1} = U_M - U_{M_s}$$

and where Ω_M , i_M , and U_M are the variables transformed to equatorial plane reference, are single valued functions of Ω_{M_s} . An IBM 1620 FORTRAN program was written which calculated Ω_M , i_M , U_{M_1} and $dU_{M_1}/d\Omega_{M_s}$ as function tables for one revolution of Ω_{M_s} . These tables were used to develop the approximations for these variables used in the analog mechanization. Since Ω_{M_s} varies by only -19.3° per year, it was possible to use linear approximations for the functions, with a maximum error of $.14^\circ$, in Ω_M

$$\Delta\Omega_{M_s} = 28K_5T_M + K_5T_D + K_5t_D/24 \quad (15)$$

$$i_M = i_{M_0} + K_{i_M} \Delta\Omega_{M_s} \quad (16)$$

$$\Omega_M = \Omega_{M_0} + K_{\Omega_M} \Delta\Omega_{M_s} \quad (17)$$

$$\Omega_{M_1} = \Omega_{M_{10}} + K_{\Omega_{M_1}} \Delta\Omega_{M_s} \quad (18)$$

$$M_M = \Omega_{M_1} + M_{M_s} \quad (19)$$

$$= K_6 + (28K_7 - 360^\circ)T_M + K_7T_D + K_7t_D/24$$

It is obvious that the initialization variables, especially Ω and M_M , will sweep through more than one cycle during the advance of injection time. Those variables which may exceed $\pm 180^\circ$ are tested continuously with comparators, which send signals to the digital logic portion. The digital logic will initiate

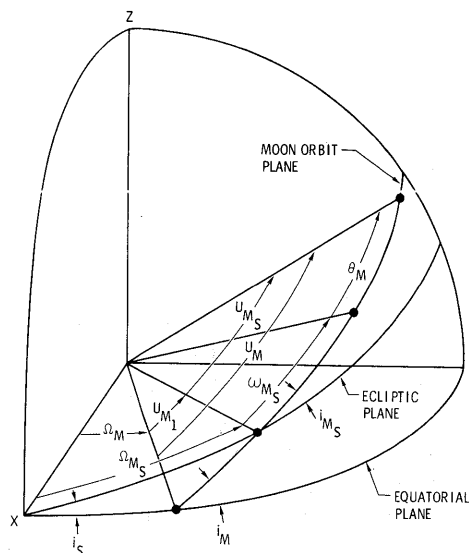


Figure 4. Equatorial, ecliptic, and moon orbit planes.

the required stop function of $\pm 360^\circ$ to prevent the variables from increasing or decreasing beyond the $\pm 180^\circ$ limits. In the case of M_M it was necessary to generate the discrete variable T_M which is incremented by one each time T_D reaches twenty-eight, and is reset. Since M_M increases at slightly more than 360° each twenty-eight days, the calculation may be done by increasing M_M by a few degrees each time T_M is incremented.

$\Delta\Omega_{M_s}$ is generated rather than Ω_{M_s} , so that we can scale the variation in Ω_{M_s} for generating the functions of Ω_{M_s} .

TRANSFORMATIONS

These transformations are used in the development of the mathematical model. In the form shown below, they are convenient collections of terms, and are useful in calculating checks since

$$\sum_{j=1}^3 X_j^2 = \sum_{j=1}^3 Y_j^2 = (A^2 + B^2 + C^2) = 1 \quad (20)$$

The transformation equations are:

$$X_{1_D} = \sin i \sin i_D + \cos i \cos i_D \cos (\Omega_D - \Omega) \quad (21)$$

$$X_{2_D} = \cos i \sin i_D - \sin i \cos i_D \cos (\Omega_D - \Omega) \quad (22)$$

$$X_{3_D} = -\cos i_D \sin (\Omega_D - \Omega) \quad (23)$$

$$Y_{1_D} = \cos i \sin (\Omega_D - \Omega) \quad (24)$$

$$Y_{2_D} = -\sin i \sin (\Omega_D - \Omega) \quad (25)$$

$$Y_{3D} = \cos(\Omega_D - \Omega) \quad (26)$$

$$A_D = X_{1D} \sin U_D + Y_{1D} \cos U_D \quad (27)$$

$$B_D = X_{2D} \sin U_D + Y_{2D} \cos U_D \quad (28)$$

$$C_D = X_{3D} \sin U_D + Y_{3D} \cos U_D \quad (29)$$

where the subscript D refers to the disturbing body (sun or moon) and unsubscripted variables refer to the satellite. Each of the terms with D subscripted variables must be generated twice, once for each extraterrestrial gravitational disturbance.

In the generation of X_j and Y_j , the approximation for i_M developed in the initialization section is used. Since the variable part of the function $i_M = i_{M_0} + \Delta i_M$ has a range of less than 4° , we expand the function of i_M by the law of sines and the law of cosines, let $\sin \Delta i_M = \Delta i_M$ and let $\cos \Delta i_M = 0$. The resulting equation for X_{1M} is:

$$X_{1M} = \sin i_{M_0} \sin i + \cos i_{M_0} \cos i \cos(\Omega_M - \Omega) + .017453 K_{i_M} \Delta \Omega_M \cos i_{M_0} \sin i - \sin i_{M_0} \cos i \cos(\Omega_M - \Omega) \quad (30)$$

The equations for X_{2M} , X_{3M} , Y_{1M} , Y_{2M} , Y_{3M} are treated in a similar manner.

The generation of X_{j_s} and Y_{j_s} is much simplified since i_s is constant and Ω_s is zero.

MEAN RATES OF CHANGE

The equations for the mean rates of change of the elements are:

$$\frac{da}{dt} = 0 \quad (31)$$

$$\frac{d\Omega}{dt} = \frac{3GM_D}{2r_D^3 n (1 - e^2)} \frac{B_D}{\sin i} \left[\frac{A_D}{2} (2 + 3e^2) - \frac{5e^2}{2} (A_D \cos 2\omega - C_D \sin 2\omega) \right] - \frac{3}{2n} J_2 \left[\frac{r_e}{a(1 - e^2)} \right]^2 \cos i \quad (32)$$

$$\frac{di}{dt} = \frac{3GM_D}{2r_D^3 n (1 - e^2)} \left[1 + \frac{3e^2}{2} C_D + \frac{5e^2}{2} (A_D \sin \omega + C_D \cos 2\omega) \right] + \frac{3}{2n} J_2 \left[\frac{r_e}{a(1 - e^2)} \right]^2 \left[\left(2 - \frac{5}{2} \sin^2 i \right) \right] \quad (33)$$

$$\frac{d\omega}{dt} = -\cos i \frac{d\Omega}{dt} + \frac{3GM_0 (1 - e^2)^{1/2}}{2r_D^3 n}$$

$$\left[-1 + \frac{3}{2} (A_D^2 + C_D^2) - \frac{5}{2} (A_D^2 - C_D^2) \cos 2\omega - 2A_D C_D \sin 2\omega \right] \quad (34)$$

$$\frac{de}{dt} = -\frac{15GM_D (1 - e^2)^{1/2}}{2r_D^3 n_1} \left[\frac{e}{2} \left[2A_D C_D \cos 2\omega + (A^2 - C^2) \sin 2\omega \right] \right] \quad (35)$$

$$\frac{dq}{dt} = -a \frac{de}{dt} \quad (36)$$

where

G = gravitational constant

M_D = mass of disturbing body (sun or moon)

r_D = distance from earth center of disturbing body

r_e = earth equatorial radius

n = mean angular motion of satellite

J_2 = coefficient of the Vinti potential function

Again the terms with a D -subscript must be computed for both solar and lunar perturbations. The terms in equations (32) and (33) with the factor $3J_2/2n$ are terms for computing the earth oblateness.⁸ The effect of earth oblateness is not significant for a , ω and e .

The equations were mechanized almost as written, with a few changes. The terms differing for solar and lunar perturbations were generated first, summed, and then multiplied by factors common to both, in order to minimize equipment use. The rate de/dt was not generated; instead, the equation was used to generate dq/dt , and e was computed algebraically after the integration of dq/dt . The reason for this was that the generation of q is of prime importance, since it is used in the success-failure test to produce the launch window, and integrating for q before calculating e places q in the feed-back loops of the mean rate equations. Therefore, any human errors made in programming the calculations of q would scarcely go unnoticed.

HIGHER ORDER TERMS

The mean rate equations shown above represent the model obtained by retaining only the first order

terms of the series expansion of the components of the perturbing forces. For high altitude satellites, this approximation is not sufficiently accurate. Therefore, some higher order terms were added to first mechanization.

P. Seery, using an IBM 7094 program developed from the same model, demonstrated that the higher order terms produced significant gains in accuracy only when included in the mean rate equation for eccentricity. While the additional terms were being scaled for analog computation, it became apparent that the higher order terms of the solar perturbation were of no significance. Therefore, only the terms for third order expansion of the lunar perturbing force were retained. The additional terms are:

$$\frac{de}{dt} = -\frac{15GM_D}{2r^3_D n} \dots + \frac{a}{r_M} \frac{1}{8} Q_{4M} (-R_{2M} - R_{3M} e^2) + \left(\frac{a}{r_M}\right)^2 \frac{7}{8} Q_{3M} Q_{4M} e (-R_{4M} - R_{5M} e^2)$$

(37)

where

$$Q_{3M} = A_M \sin \omega + C_M \cos \omega \quad (38)$$

$$Q_{4M} = A_M \cos \omega - C_M \sin \omega \quad (39)$$

$$R_{2M} = -4 + 5Q_{3M}^2 + 5Q_{4M}^2 \quad (40)$$

$$R_{3M} = -3 + 30Q_{3M}^2 + 5Q_{4M}^2 \quad (41)$$

$$R_{4M} = 6 - 7Q_{3M}^2 - 5Q_{4M}^2 \quad (42)$$

$$R_{5M} = 3 - 14Q_{3M}^2 + 7Q_{4M}^2 \quad (43)$$

Comparison of the model, with and without the addition of the higher order terms, is shown in Fig. 5.

INTEGRATIONS

The mean rates of the elements were integrated to obtain the changes from the initial values, Δi , $\Delta \omega$, $\Delta \Omega$, and Δq ; this allowed better scaling for plotting time histories of the elements, and more accurate multiplication for many of the products calculated in the mean rate equations.

The functions $\sin U_s$ and $\cos U_s$ were generated using a constant frequency sine wave generator circuit, with the integrators initialized by an electronic resolver driven by the initialization variable U_s . $\Delta \Omega_{M_s}$ was generated by integrating the mean rate (assumed constant) to produce a ramp function. The functions $\sin U_M$ and $\cos U_M$ were generated using a continuous resolver circuit, integrating the mean (constant) rate of U_M , switching the rate to produce a triangular wave as the integrator output, driving an electronic

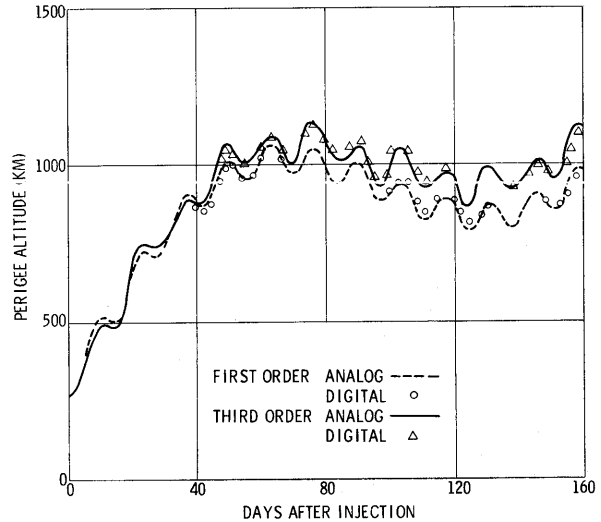


Figure 5. First and third order models.

resolver with the triangular wave, and using the same switching signal to select the proper sign of $\sin U_M$.

DIGITAL LOGIC

When plotting a launch window, hour of the day (t_D) is used to drive the pen of a standard X-Y plotter. The response of a plotter is too slow to follow this signal if t_D is reset to zero at the end of each twenty four hour period. To eliminate this problem t_D was generated as a series of discrete steps from zero to twenty four hours and then from twenty four back to zero. The circuitry used is shown in Fig. 6. The primary control amplifier in this operation is number 30 since it controls the step size and the sign of the step. Amplifier 30 is a standard analog integrator used as a three level switch. In this application, the capacitor is not bottle plugged to the grid, and the grid is used as an input from an external summing network. During operation, the output of this integrator will be $(t_D - \Delta t_D)$ when the hold relay is de-energized or $(t_D + \Delta t_D)$ when the IC relay is de-energized. If both the hold and IC relays are energized, the output will be the voltage present at the IC input, in this case zero. The state of the mode control relays is controlled by flip flop "B" and the mode control push buttons. Amplifiers 25 and 26 are standard integrators with electronic mode control and are used as track and hold amplifiers. If the computer is in IC by means of the mode control push buttons, amplifiers 25 and 26 are in reset and are tracking the output of amplifier 30. Both the IC and hold coil of integrator 30 are

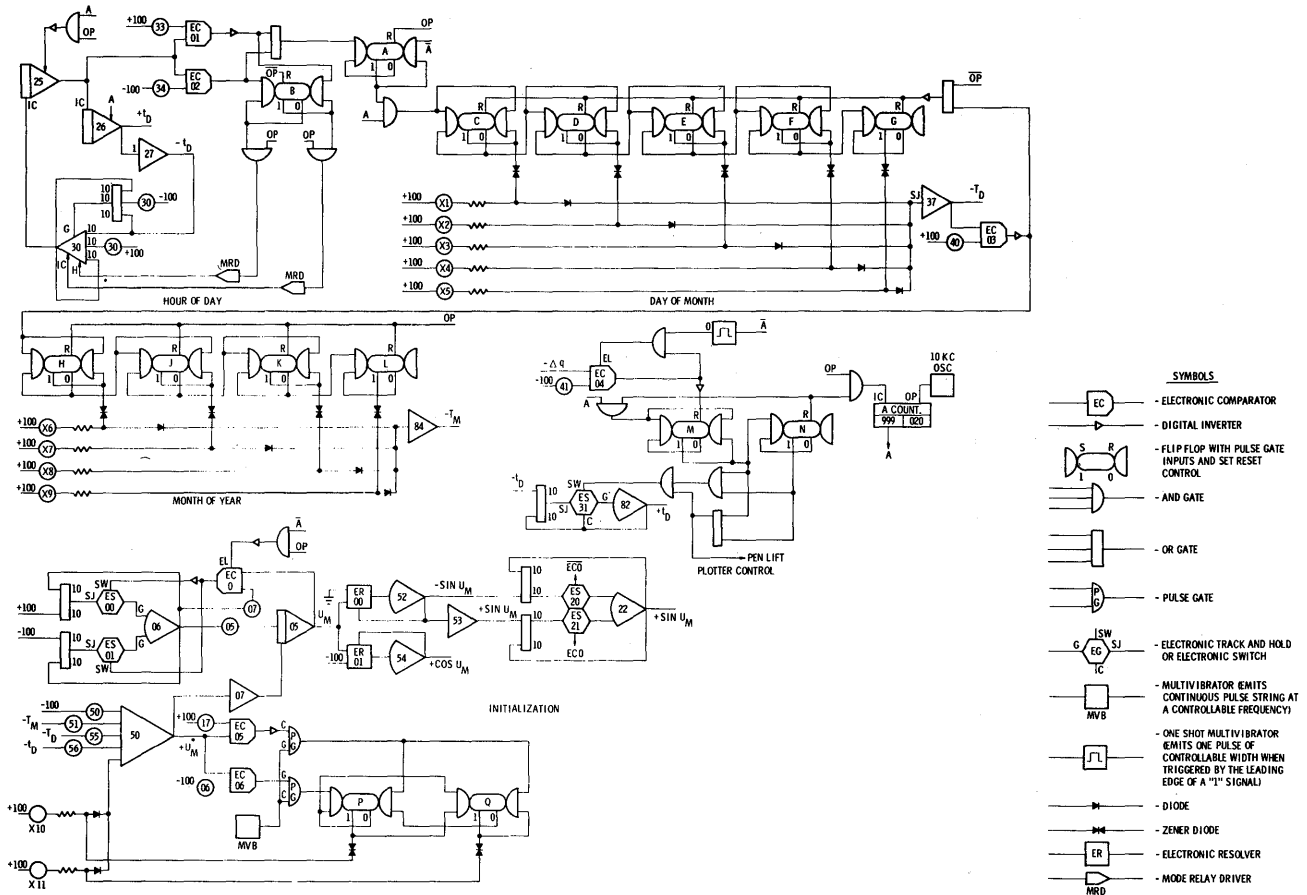


Figure 6. Digital logic.

energized so that its output is zero at this time. When the operate button is depressed, amplifiers 25 and 26 are under the control of the rep op counter so that 25 is in the track mode and 26 in hold during the operate cycle and the IC relay of integrator 30 is de-energized so that its output is equal to the output of 25 plus one time step. During the reset cycle of the "rep on" control, amplifier 25 is in hold and 26 in track. Operating in this fashion the output of amplifier 25 will be updated during the reset cycle by one time increment until amplifier 25 is equal to twenty four hours plus one time increment. At this time flip flop "B" is reset by electronic comparator 01 and the IC coil of integrator 30 is energized and the hold coil de-energized, and the output of 25 steps back down from twenty-four to zero at which time comparator 02 again reverses the operation. The output of amplifier 25 appears as a quantized triangular wave until the IC control button is depressed.

A standard binary up counter was used to count the days of the month and month of the year (Fig.

6). At the end of each 24 hour period, or when comparator 01 or 02 was high, flip flop "A" was set so that on the next reset cycle of the "rep op" mode control the day counter would be incremented by one. When the output of amplifier 37 was equal to the twenty ninth day comparator 03 reset the day counter and incremented the month counter. The output of the digital counters was used to control diode gates on the external gains of pots X1 through X9. The external pots were set so that, when they were turned on, the output of amplifiers 37 and 84 was a D.C. voltage equal to the digital value of the binary counter. Since the Harman Hardon digital logic operates on a zero minus twelve volt logic level, a zener diode was required in the diode switches to allow the positive voltage to pass when the switch is turned on.

As previously mentioned, some initial conditions change greater than $\pm 180^\circ$ during a normal running time, and in order to remain within the normal operating range of the electronic resolvers, special switching logic must be used.

An example of the logic used is shown in Fig. 6 in generating U_M . Each time U_M becomes equal to $+180^\circ$ flip flops P or Q will be set and 360° subtracted from U_M . At -180° the flip flops are reset, and the diode gates turned off. To eliminate the possibility of the logic being in the wrong state, especially when the computer is reset, a multivibrator operating at about 1 K.C. was used to pulse the flip flops if U_M was greater than $\pm 180^\circ$. Also shown is the triangular wave generator used for U_M and the switch required to invert the sign of the sine U_M each time U_M goes through 180° . Since the launch window only reflected whether a particular injection time resulted in a successful or unsuccessful orbit considerable computation time was saved by resetting the computer immediately upon a failure. In the case where the failure occurred early in the year several milliseconds could be saved per computational cycle. The net result was a savings of several minutes per launch window.

RESULTS

Launch windows were plotted with the computers in high-speed repetitive operation mode, with the time scale of one year equal to .1 second computer time. Each plot was made for a period of 120 days with a mesh of one day by .2 hours, so that each launch window required 14,400 analog runs. The time required to produce one launch window ranged between twenty to twenty five minutes. In addition, time histories of perigee altitude were made for approximately sixty different injection times, with the solution speed slowed by a factor of 100, as supporting data for each of several of the launch window plots.

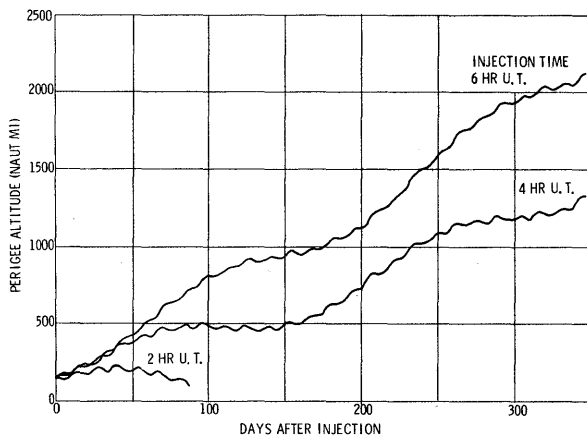


Figure 7. Perigee time histories.

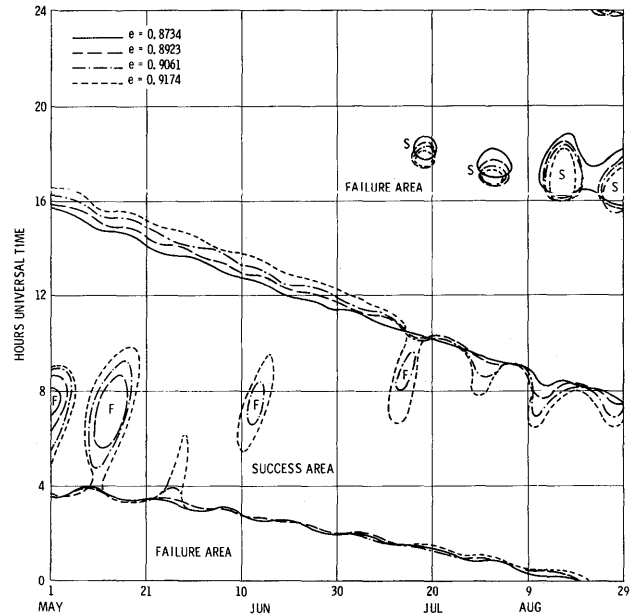


Figure 8. Family of launch windows.

The present model showed a significant increase in accuracy over models previously used at Martin-Baltimore in similar programs; and the comparison with digital runs made by P. Seery was very good, as shown in Fig. 5. Figure 7 shows some typical perigee time histories, illustrating the effect of changing the hour of injection time during the same day. Figure 8 shows a family of launch windows, combining four plots to show the effect of increasing the semi-major axis and initial eccentricity, while initial perigee, altitude, inclination, and right ascending node remain the same. The effect of the lunar periodic perturbation can be observed as a slight variation about the general trend of the boundary, with a period of one half of the lunar period of revolution. This effect is shown to increase as eccentricity (hence apogee altitude) increases. Also, it shows the appearance of "islands" of failure within the success area as eccentricity increases, which connect to the peaks of the boundary variation as eccentricity is increased. Another trend shown is the widening of the success area at the upper boundary, indicating the increased effect of the lunar secular perturbation.

Launch windows, generated by the digital methods previously used, have required approximately 35 hours of IBM 7094-II time and three months elapsed time for the generation of one plot. Furthermore, it was customary to cover the region with an extremely coarse mesh, eliminate large sections of the region shown not to contain a segment of the principal

boundary, and cover the remaining areas with a finer mesh. Since the total area is not covered by a uniform mesh, as it is on the analog program, there is a possibility that islands of failure within a large success area may not be discovered.

In contrast, the analog program requires about three hours to set up and check out, with about one-half hour for each launch window, and about one hour for each series of time histories.

The present computer program was used to generate twenty-one launch window plots, and eight series of associated time histories, at a cost of approximately \$2500. The estimated cost for acquiring the same information by digital methods is about \$500,000. The elapsed time for the 21 analog launch windows was one week; the minimum elapsed time for 21 digital launch windows would be about one year.

CONCLUSION

The significance of the cost and elapsed time savings of this method is the increased amount of information available for the planning of a satellite program. Using digital methods, only a few such plots could be generated within the applicable cost and time constraints, and the choice of those few had to be made from the many possible before the informa-

tion was acquired. Now it is feasible to investigate a wide range of nominal orbits, together with statistically probable variations in the nominal orbits, early enough to be useful in the satellite program.

REFERENCES

1. F. R. Moulton, *An Introduction to Celestial Mechanics*, Macmillan Company, New York, 1956.
2. J. L. Stricker and W. W. Miessner, "Launch Window Program," Analog Problem 702, Martin-Baltimore.
3. ———, "Satellite Orbital Stability Program," *Proceedings of the IFIP Congress, Volume 2*, Spartan Books, Washington, D.C., 1966, p. 405.
4. ———, "Launch Window Program," to be published in *SIMULATION*.
5. M. M. Moe, "Solar Lunar Perturbations of the Orbit of an Earth Satellite," *American Rocket Society Journal* (May 1960).
6. D. E. Smith, "The Perturbation of Satellite Orbits by Extraterrestrial Gravitation," *Planetary Space Sciences*, vol. 9 (1962).
7. J. Glazer and C. Bowie, "Definite Integrals and Average Changes used in the Application of Analogue Equipment to Smith's Satellite Equations," Martin-Baltimore Inter-Department Communication, 8 September 1965.
8. G. E. Townsend, Jr., et al, "Orbital Flight Handbook," Martin-Baltimore.

TRAJECTORY OPTIMIZATION USING FAST-TIME REPETITIVE COMPUTATION

Rodney C. Wingrove

and

James S. Raby

Ames Research Center, NASA, Moffett Field, Calif.

INTRODUCTION

Space vehicle trajectories must be near optimum in the sense that some parameter is either a maximum or a minimum; for example, in reentry the trajectory to desired terminal conditions is near optimum when the total aerodynamic heating is a minimum. Several perturbation methods,¹ such as the calculus of variations, applications of the maximum principle, and direct steepest descent, have been considered for determining the time histories of nonlinear controls that correspond to optimum trajectories.

The computations² in these previous optimization studies involved the dynamic solution of two sets of equations: (1) nonlinear state equations; and (2) linear adjoint equations. An alternate perturbation computation technique—the impulse response method³—will be discussed here. This method differs from previous studies in that only the solution of the nonlinear state equations is used. The response of given functions (e.g., terminal error or quantity to be optimized) to a control impulse is determined along the trajectory by fast-time repetitive computations rather than by a solution of adjoint equations. This impulse response method enables the investigator to retain an intuitive understanding of the

optimization process. Furthermore, since adjoint equations are not required, the state equations or cost functions need not be amenable to linearization. The impulse response method does require many solutions of the state equations; however, the programming is straightforward and the task of computing a large number of dynamic solutions is ideally suited to modern high-speed computers.

NOTATION

The following notation is used in the body of the text. Additional symbols are described as they are introduced.

L/D	control value of lift-drag ratio
n	number of storage points in control time history
t	time
t_f	final time
t_0	initial time
Δt	time increment of control impulse
u	control
Δu	height of control impulse
φ	cost at final time
$\Delta\varphi(t)$	change in cost at final time due to control impulses at time t

ψ	state value at final time
ψ_d	desired state value at final time
$\Delta\psi(t)$	change in state value at final time due to control impulses at time t

GENERAL OUTLINE OF METHOD

The impulse response method, as discussed in this report, uses the steepest descent optimization process.⁴⁻⁷ The process commences with any nonoptimal trajectory from which a slightly improved one is derived. The improved trajectory is then used as a new nominal trajectory, and the procedure is repeated until the optimum or nearly optimum trajectory is found.

The iterative procedure is: (1) estimate a reasonable nominal control program; (2) determine impulse response functions that indicate the best method of making small changes in the control that will decrease the cost (the quantity to be minimized); (3) compute a new nominal control by adding this change in control to the previous nominal control (this results in a new trajectory with a decreased cost); (4) repeat step 2. This iterative process continues until the change in cost for each new trajectory is very small; the control is then very near a local optimum. If at any point along the trajectory a limit value of the control is reached before the cost is completely minimized, no further optimization is possible at that point. In this case, the process continues until at each point on the trajectory either a local optimum or the control limit is reached.

Computation of Impulse Response Functions

The technique by which the impulse response function is determined is the most important feature of the impulse response method. Figure 1 illustrates the manner in which the influence of small control changes on the cost is calculated. The equations of motion are first solved with a positive control impulse at time t superimposed upon the nominal control. During the next solution of the equations of motion, a negative control impulse of the same magnitude is inserted at time t . The impulse response, $\Delta\varphi$, is derived from these two solutions. In a similar manner, the impulse response is determined at successive times along the trajectory. The impulse response function, $\Delta\varphi(t)$, is the complete time history of $\Delta\varphi$. Its computation for the same control impulse at different times along the trajectory is defined as one iteration. This corresponds to previous

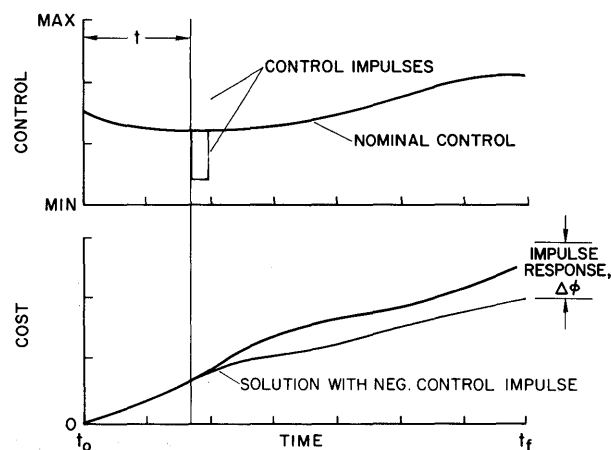


Figure 1. Computation of the impulse response.

optimization studies⁴⁻⁷ that used one iteration of the adjoint equations to compute essentially the same impulse response function along the trajectory.

Calculation of Minimum Cost

When the cost is to be minimized and there is no terminal constraint, the impulse response function is used in the steepest descent technique to modify the control toward the optimum in the following manner:

$$\begin{pmatrix} \text{New} \\ \text{Nominal} \\ \text{Control} \end{pmatrix} = \begin{pmatrix} \text{Previous} \\ \text{Nominal} \\ \text{Control} \end{pmatrix} + K_{\varphi} \Delta\varphi(t) \quad (1)$$

The gain K_{φ} weights the impulse response function for the cost; its sign is negative to decrease the cost. The magnitude of K_{φ} is determined experimentally for each problem: too large a gain may cause instability in the convergence procedure, while too small a gain may extend the time of convergence.

A representative sample of what one may expect with this type of optimization procedure is sketched in Fig. 2. During the first iteration, the repetitive solutions determine the impulse response function, $\Delta\varphi(t)$. This $\Delta\varphi(t)$ is added to the nominal control with an appropriate gain K_{φ} , and the new nominal control time history, as shown in the center of Fig. 2, is obtained. The iterative process is repeated until the optimum control is reached. The optimum control may take on either or both of the properties illustrated in the final iteration of Fig. 2. In the region (A) the impulse response function is, for all practical purposes, zero. This implies that a small change in control in this region will not modify the cost; thus the control is at a local optimum. In region (B) the control is at the limiting constraints,

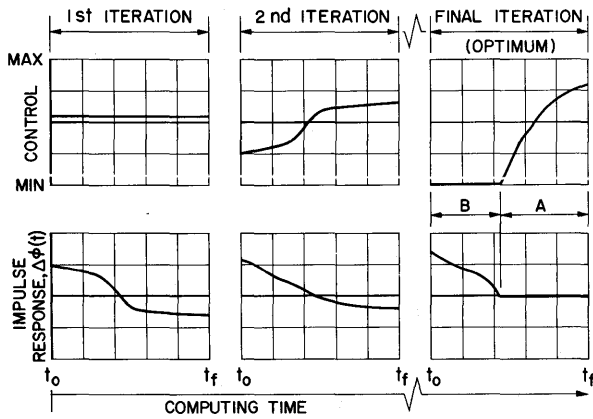


Figure 2. Computation of the optimal control.

and the impulse response function indicates that only control beyond the constraint will decrease the cost. Thus, on the constraint, the control is at a local optimum.

Minimum Cost With Terminal Constraint

When a terminal constraint (or destination) is to be reached, while minimizing the cost, the iteration procedure is performed as follows:

$$\begin{pmatrix} \text{New} \\ \text{Nominal} \\ \text{Control} \end{pmatrix} = \begin{pmatrix} \text{Previous} \\ \text{Nominal} \\ \text{Control} \end{pmatrix} + K_{\varphi} \Delta\varphi\Delta(t) + K_{\psi} \Delta\psi(t) \quad (2)$$

where ψ denotes the state variable at the final time. The quantity $\Delta\psi(t)$ represents the change in the state variable due to control impulses, and is evaluated in the same manner as $\Delta\varphi(t)$.

Gains K_{φ} and K_{ψ} are constants for each iteration. Gain K_{φ} weights the impulse response function for cost; its sign is negative to decrease the cost. Gain K_{ψ} must be calculated for each iteration so that the term $K_{\psi} \Delta\psi(t)$ will account for terminal displacement due to the optimizing term, $K_{\varphi} \Delta\varphi(t)$, and correct any terminal displacement error from the previous iteration.

The equation for K_{ψ} is:

$$K_{\psi} = -K_{\varphi} \frac{\int_{t_0}^{t_f} \Delta\varphi(t)\Delta\psi(t) dt}{\underbrace{\int_{t_0}^{t_f} \Delta\psi^2(t) dt}_{\text{Steepest descent optimization term}}} + 2 \Delta u \Delta t \frac{\psi_a - \psi}{\underbrace{\int_{t_0}^{t_f} \Delta\psi^2(t) dt}_{\text{Terminal error correction term}}} \quad (3)$$

The derivation of this equation can be found in Appendix A. Δu is the height of each control impulse; Δt is the time interval of each control impulse; ψ_a is the desired end-point value; and $\psi_a - \psi$ represents any terminal displacement error from each previous iteration. This equation gives the general form of the steepest descent computations; the computer mechanization of this method will be discussed next.

COMPUTER MECHANIZATION AND RESULTS

The impulse response method has been mechanized on both a hybrid and a digital computer to determine the optimal time history of the lift-drag ratio (control L/D) that must be flown for a vehicle returning into the earth's atmosphere. The example problem requires that: (1) the cost, φ , which is the heat input to the vehicle, be minimized; (2) the vehicle arrive at a terminal constraint, ψ_a , (destination); and (3) the control time history remain within specified limits. The solution to this particular problem is known a priori to be a bang-bang control; therefore, the final results can be verified.

The equations of motion are presented in Appendix B. The vehicle characteristics and flight conditions were those of a manned capsule returning from earth orbit and having the following parameters:

Initial conditions:	altitude	250,000 ft
	horizontal velocity	25,000 fps
	vertical velocity	748 fps
	range to destination	1,000 miles
Stopping conditions:	altitude	100,000 ft
Control limits:	L/D	$0 \leq L/D \leq 0.5$

The computer systems used in the two mechanizations are described in appendix C.

Hybrid Computer Mechanization

The major elements of the hybrid computer consisted of: (1) an analog computer to solve the trajectory equations; (2) parallel digital logic units to control the computer program; (3) delay line memories to store the control time history; and (4) D-A and A-D converters to transfer the control time history between the analog computer and the delay line memory.

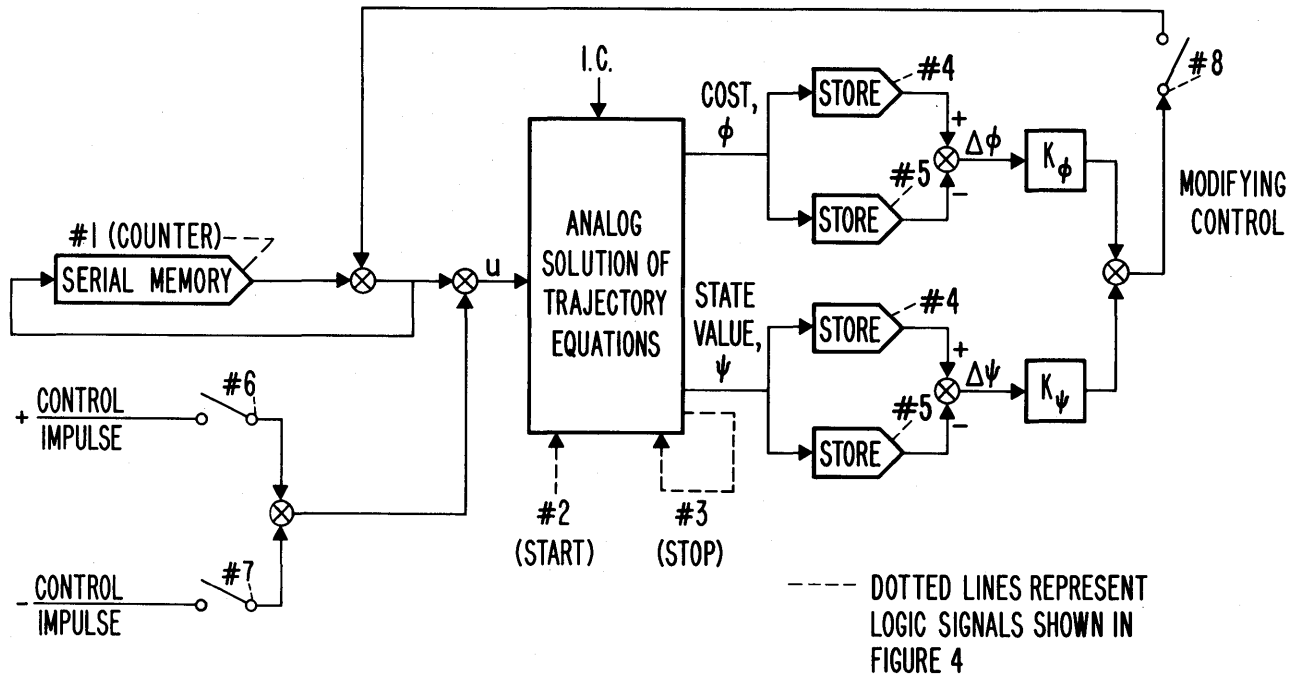


Figure 3. Hybrid computer flow diagram.

The L/D time history was stored in 64 word serial delay line memories with a resolution of 13 bits. The access time of the serial memory was 128 μsec . To permit a complete solution of the trajectory equations within the 128 μsec , the analog computer was time-scaled at 3750 to 1.

The mechanization of this problem on the hybrid computer is illustrated in Figs. 3 and 4: Figure 3 is the problem flow chart, and Fig. 4 illustrates the logic used in controlling the problem. The serial memory unit is continuously driven by counter pulses (Logic No. 1). The output of the serial memory is the nominal control time history with n points. This time history is used, together with the appropriate control impulse, to solve the trajectory equations. These equations are started at the specified initial conditions with Logic No. 2, and stopped with Logic No. 3 when the trajectory reaches the specified end condition on altitude. The final values of the cost quantity (heat) and the state quantity (range) are stored at the end of each run as indicated by Logic Nos. 4 and 5. The positive or negative control impulse is added to the nominal control input with Logic Nos. 6 and 7, respectively. Logic No. 8 inserts the modifying control ($K_\phi \Delta\phi(t) + K_\psi \Delta\psi(t)$) into the serial memory. This procedure runs in essentially a continuous manner, that is, one point out of the

n points in the nominal control history is updated after each two repetitive computations. After $2n$ repetitive computations (one iteration), every point in storage has been modified and the process is repeated. For each iteration, gains K_ϕ and K_ψ are held constant. As previously mentioned, gain K_ϕ determines the relative speed and stability of the convergence onto the optimum. The corresponding value of K_ψ to be used with each new iteration is calculated by Eq. (3) as a function of the terminal error from each previous iteration ($\psi_d - \psi$) and the following

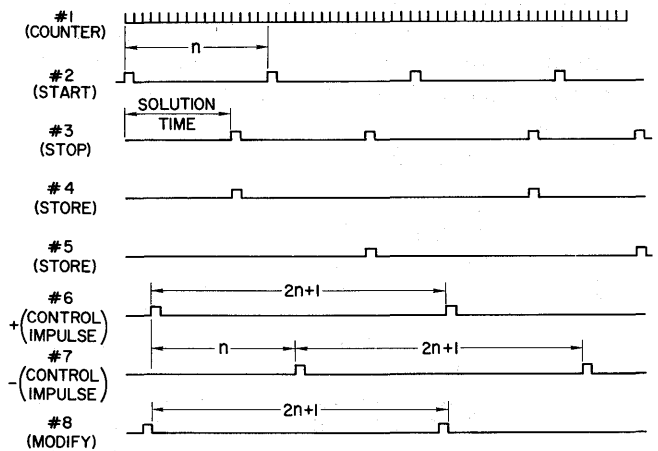


Figure 4. Hybrid computer program logic.

two integrated quantities from each previous iteration:

$$\int_{t_0}^{t_f} \Delta\varphi(t)\Delta\psi(t)dt \quad (4)$$

and

$$\int_{t_0}^{t_f} \Delta\psi^2(t)dt \quad (5)$$

Time t_0 was represented by a logic signal at the first repetitive computation in an iteration cycle and time t_f was represented by a logic signal at the last computation in an iteration cycle. It should be noted that during those parts of the trajectory when the control was at a constraint limit, no further optimization was possible and the integration of Eqs. (4) and (5) was not carried out during those times.

Hybrid Computer Results

The results obtained from the hybrid simulation are illustrated in Figs. 5 and 6. Figure 5 shows a portion of one iteration, while Fig. 6 shows the convergence to the optimum control L/D .

In the upper trace of Fig. 5, the control impulses are superimposed upon the initial nominal control. Each control impulse had a magnitude of $L/D = \pm 0.25$ and a time increment of one clock pulse (0.002 sec). This control impulse was chosen because it gave variation in the final range and heat load on the order of ± 5 percent. The integrated heat loads along each of the repetitive trajectories are presented in the next trace. The difference between the final quantities for each pair of subsequent runs is $\Delta\phi$, and represents the heat load impulse response.

In Fig. 6, the first few iterations of the converging optimization procedure are illustrated together with

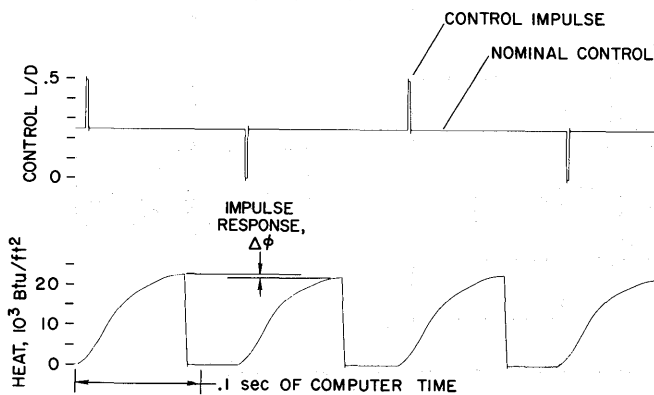


Figure 5. Hybrid repetitive computations.

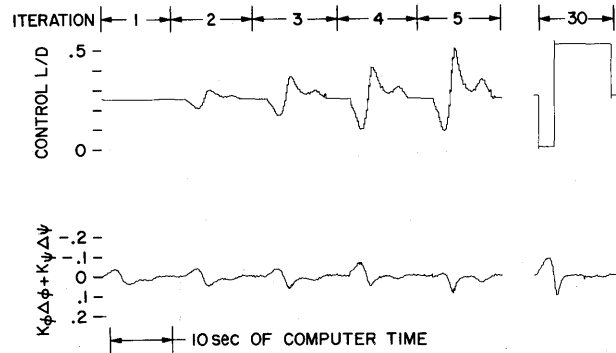


Figure 6. Hybrid computation of the optimal control.

the thirtieth iteration. In the upper trace the nominal control is recorded as it is read out of serial memory every $128 + 1$ counter pulse (with Logic No. 8). This gives a convenient time history to show the manner in which the control has been modified during each iteration. Notice that the control is limited within $0 \leq L/D \leq 0.5$. This was achieved by simply limiting the output of the serial memory to these values. The modifying control shown in the lower trace of Fig. 6 is the sum, $K_\varphi \Delta\varphi(t) + K_\psi \Delta\psi(t)$. For this series of runs, a constant $K_\varphi = -2.5 \times 10^{-3}$ / Btu/ft² permitted fairly rapid convergence while program stability was maintained. The value of K_ψ was calculated for each iteration by Eq. (3) to be that value which kept the final value of range near 1,000 miles.

As can be seen in Fig. 6, the optimum control variation for this particular example was a bang-bang control. With the steepest descent method, it was found that near-optimum control could be achieved in the first few iterations, but to "square up the corner" and achieve full optimum control required more iterations (20 to 30).

Digital Computer Mechanization

The major elements of the digital computer system consisted of: (1) a digital computer to solve the trajectory equations, perform the logical control of the program, and store the control L/D time histories; (2) a line printer to print hard copies of the results; and (3) D-A converters and a strip chart recorder for fast observation of trends.

The digital program was written in floating point symbolic language. Since the optimization technique requires repetitive computation of the trajectory, the choice of an integration routine was very important. A fast, stable, and fairly accurate routine was

needed. These requirements conflict to some extent;^{8,9} however, the fourth-order Adams-Bashford integration algorithm gave satisfactory results at a step size of 5 seconds, provided a satisfactory starter was used. The starter used the lower order Adams-Bashford algorithms with a step size of 1 second.

The program flow is as follows (see Fig. 7): (1) A nominal control time history is used to calculate the nominal trajectory; (2) this trajectory is stored for use as the initial conditions for the repetitive computations of the trajectory; (3) at the initial point along the nominal trajectory, the control is perturbed with a positive pulse, and a new trajectory is calculated; (4) at this same point on the trajectory, the control is perturbed with a negative pulse and another new trajectory calculated; (5) from these two repetitive computations of the trajectory the heat impulse response, $\Delta\phi$, and the range impulse response, $\Delta\psi$, are calculated; (6) the program is then advanced to new initial conditions along the nominal trajectory by the length of the integration step size; (7) steps (3) through (6) are repeated until the

initial altitude reaches the stopping condition (100,000 ft); (8) at this time, a new nominal control time history is computed using Eqs. (2) and (3); (9) steps (1) through (8) are repeated. This iterative computation continues until an optimum trajectory is reached.

Digital Computer Results

The first five iterations and the twentieth iteration of the digital simulation are illustrated in Fig. 8. The upper trace of Fig. 8 shows the control L/D time history. During the first iteration, the control L/D was a constant 0.25; at the end of this iteration it was modified by Eqs. (2) and (3). By the fifth iteration the control L/D was approaching bang-bang and by the twentieth iteration it was essentially bang-bang. The pulse used to perturb the trajectory had a height of 0.25 L/D and a width equal to one integration step size. For this pulse, a constant value of $K_\phi = -7.5 \times 10^{-2} / \text{Btu}/\text{ft}^2$ permitted a fairly rapid convergence and the computation remained quite stable.

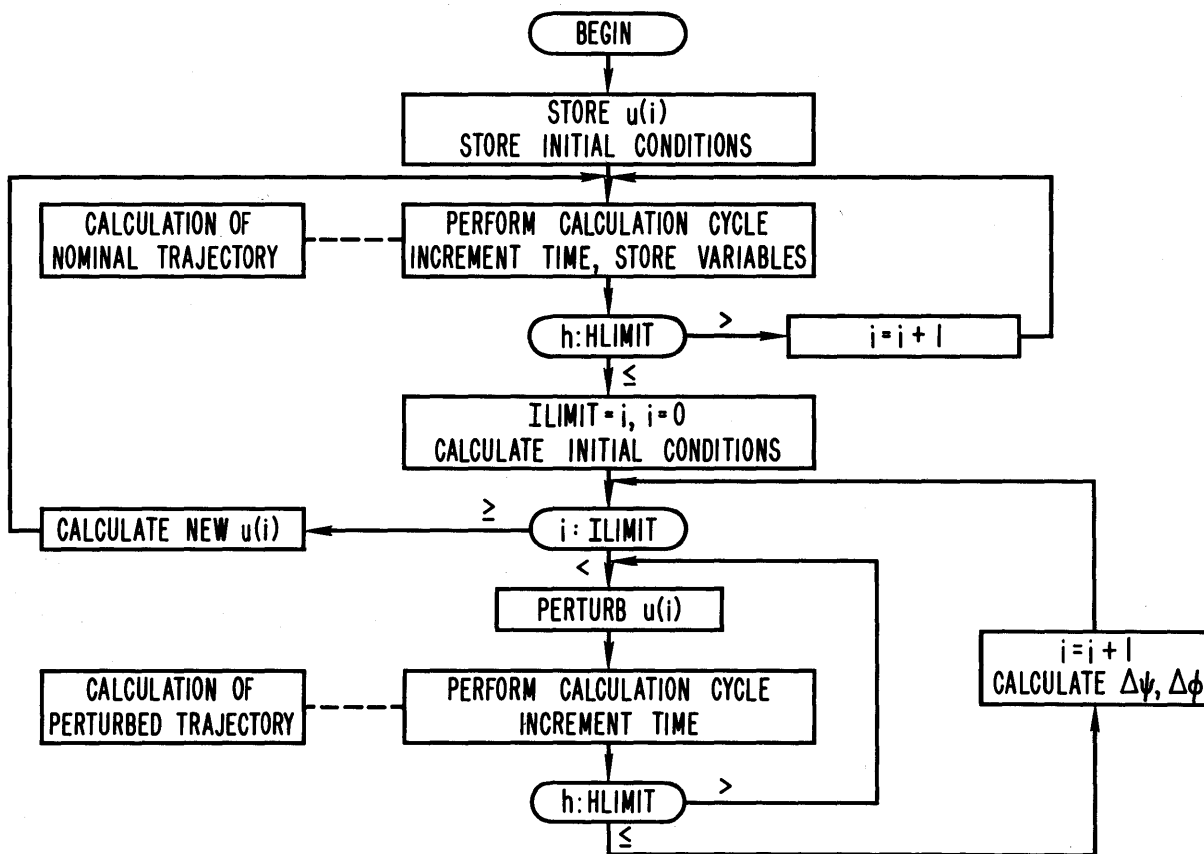


Figure 7. Digital computer flow chart.

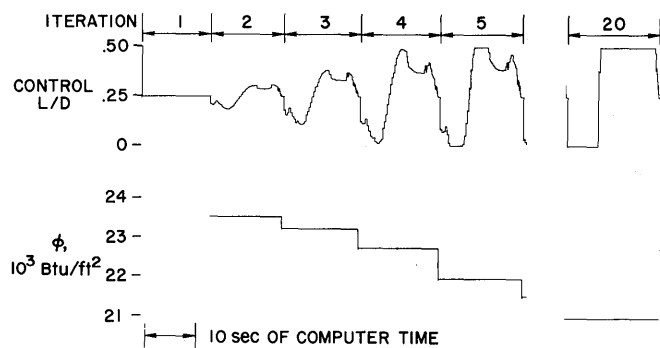


Figure 8. Digital computation of the optimal control.

The second trace of Fig. 8 shows the variation in heat from one iteration to the next. The heat, which is the cost in this example, decreases markedly during the first five iterations and nearly reaches its final value by the end of the fifth iteration. Tables I, II, and III give the results in tabular form. The range is shown to remain near 1,000 miles while the heat is reduced from 23,491 Btu/ft², at the end of iteration 1, to 21,517 Btu/ft² at the end of iteration 5. The major change during iterations 5 through 20 was to "square up" the *L/D* control and achieve the full optimum control. At the end of iteration 20, the final range achieved was 999.9 miles and the heat 20,966 Btu/ft². During the optimization procedure the range varied slightly about the desired value of 1,000 miles and the heat load was reduced about 10 percent.

Discussion of Hybrid and Digital Results

It was interesting to observe that both the hybrid and the digital simulations required approximately

Table I.—Altitude Time Histories

Time, sec	Altitude, 10 ³ ft			
	Iteration			
	1	2	10	20
0	250	250	250	250
60	207	206	197	197
120	180	184	160	160
180	182	184	212	209
240	158	165	196	195
300	125	132	142	142
360			127	127
400			104	110

Table II.—Control Time Histories

Time, sec	Control <i>L/D</i>			
	Iteration			
	1	2	10	20
0	0.250	0.212	0	0
60	.250	.192	0	0
120	.250	.291	.500	.500
180	.250	.290	.500	.500
240	.250	.294	.447	.500
300	.250	.294	.500	.500
360		.250	.347	.475
400			.254	.277

the same amount of computer time, approximately 2 minutes to obtain near optimum trajectories and approximately 5 minutes to obtain full optimum trajectories. However, it should be pointed out that no real attempt was made to minimize either of these computing times. There are several methods for reducing the computer time required to obtain optimum trajectories. One method would be to select the gain *K_φ* automatically for each iteration instead of using a constant value for the entire computing run. This would cause the solution to converge to an optimum in fewer iterations at the expense of complicating the computer program. Another method of decreasing the computation time would be to decrease the number of points used to store the control time history which would decrease the number of repetitive computations required for each iteration.

The results obtained by both the hybrid and the digital computer appear satisfactory for engineering purposes. The final values of range and heat computed by the two simulations agree to within approximately one percent and both simulations arrived at the same bang-bang control time histories.

Table III.—Terminal Conditions

	Iteration				
	1	2	5	10	20
Time, sec	344	358	389	407	414
Altitude, 10 ³ ft	99.3	99.9	99.8	99.5	98.8
Range, miles	997.7	1001.5	1003.7	1002.8	999.9
Heat, Btu/ft ²	23491	23197	21517	21025	20966

One excellent feature of the digital simulation was the program documentation obtained by using the on-line typewriter and line printer. The typewriter documented every change made during the time the program was in the computer, and the line printer permitted the analysis of each variable at specific points along the trajectory. Equally valuable was the strip chart recording normally obtained in hybrid computation. It was obtained in the digital program by D-A conversion of the digital variables. This "quick look" capability made it possible to observe trends not readily apparent in numerical printouts.

The result of this test example was no surprise. In simulations that require complicated logic control of the program and a moderate amount of storage, there is a distinct advantage to using a digital computer. It proved reasonable to use a digital computer in this simulation because there was only a moderate number of simplified equations to be solved. If the number of equations were increased, the time to solve them on the digital computer would, of course, also increase.

COMPARISON WITH ADJOINT STEEPEST DESCENT

A current reentry optimization study at Ames Research Center is using both the impulse response method of this report and the standard adjoint steepest descent computing method. This study is of interest because the two methods have been programmed on the same computer (IBM 7094) and their ability to solve several identical problems has been compared.

Representative solutions obtained from the two methods are illustrated in Fig. 9. This particular example is for the same reentry vehicle and initial flight conditions used in the previous example of this report. However, the cost function is of the form:

$$\varphi = \int_{t_0}^{t_f} [\text{Heat rate} + (\text{Drag})^2] dt$$

and there is no terminal constraint. This was chosen in order to illustrate a problem formulation that does not represent a bang-bang optimal control result.

The results of the twentieth iteration are shown in Fig. 9. The upper curve shows that the control solutions are almost identical. In the lower curve the impulse response function $\Delta\varphi(t)$ has been normalized³ for comparison with the corresponding results

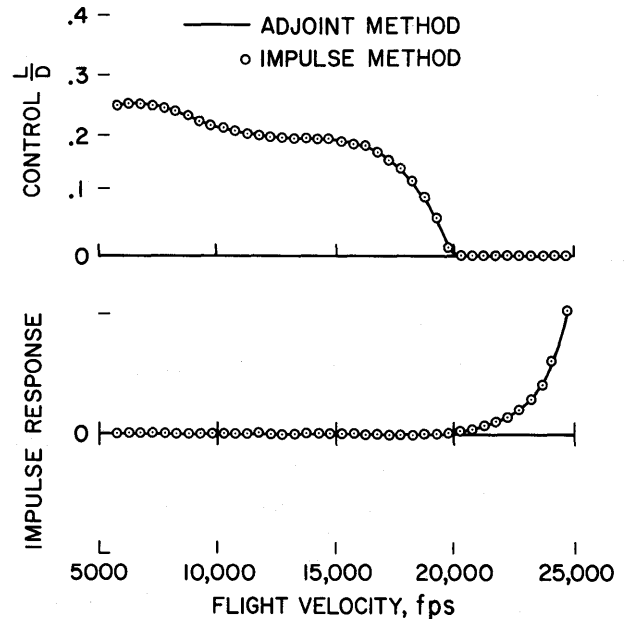


Figure 9. Comparison of the impulse response and adjoint steepest descent methods.

obtained by the adjoint solutions. Figure 9 demonstrates that the two methods arrive at essentially the same final solution.

For reentry problems similar to the one presented herein, it has been found that the computing time required with the adjoint method is about one order of magnitude less than that required by the impulse response method. Because the adjoint method uses less computer time, it has been the more desirable method for production runs that require a large number of optimized trajectories. However, because the impulse method is straightforward to program and because the engineer is able to retain an intuitive understanding of the optimization procedure, the impulse method has been the more desirable method for initial problem mechanization. Furthermore, adjoint equations require linearization and, therefore, cannot be used in some problem formulations. For example, in reentry problem formulations with complicated heat-balance equations,¹⁰ rather than the simple heating expression shown in appendix B, the heat rate cannot be linearized. In this type of formulation, the impulse response method has provided the only practical solution.*

* Dynamic programming was also tried for this problem but the computer time was found to be excessive, one to two orders of magnitude greater than that required with the impulse response method.

CONCLUSIONS

This paper has described reentry trajectory optimization using the impulse response method. The method requires that the computer perform a large number of fast-time repetitive computations in solving the state equations and in determining impulse response functions. These repetitive computations are readily performed by both hybrid and digital computers.

The mechanization of the impulse response method on both hybrid and digital computers was found to be straightforward. Near optimum reentry trajectories were obtained in approximately 2 minutes and full optimum reentry trajectories in approximately 5 minutes of computer time. The solutions obtained from either mechanization agreed to within approximately one percent.

The impulse response method has been compared with the adjoint steepest descent method. The solutions obtained by either method were essentially identical. The adjoint method requires less computer time; however, the impulse response method does not require familiarization with or use of an auxiliary set of linear adjoint equations. Furthermore, for problem formulations that are not amenable to linearization, the impulse response method may be the only practical method.

APPENDIX A

DERIVATION OF EQUATION FOR K_ψ

Along a normal trajectory, small changes, $\delta\psi$, in the terminal state due to small changes, $\delta u(t)$, in control can be approximated by:

$$\delta\psi = \frac{1}{2 \Delta u \Delta t} \int_{t_0}^{t_f} \delta u(t) \Delta\psi(t) dt \quad (A1)$$

where Δu is the height of each control impulse and Δt is the time interval of each control impulse. Substituting $K_\varphi \Delta\varphi(t) + K_\psi \Delta\psi(t)$ from Eq. (1) for $\delta u(t)$, we have:

$$\delta\psi = \frac{1}{2 \Delta u \Delta t} \int_{t_0}^{t_f} [K_\varphi \Delta\varphi(t) \Delta\psi(t) + K_\psi \Delta\psi^2(t)] dt \quad (A2)$$

Solving for K_ψ and letting $-\delta\psi = \psi_a - \psi$ (the previous terminal error), we obtain:

$$K_\psi = -K_\varphi \times$$

$$\frac{\int_{t_0}^{t_f} \Delta\varphi(t) \Delta\psi(t) dt}{\int_{t_0}^{t_f} \Delta\psi^2(t) dt} + 2 \Delta u \Delta t \frac{\psi_a - \psi}{\int_{t_0}^{t_f} \Delta\psi^2(t) dt}$$

Steepest descent optimization term
Terminal error correction term

(A3)

APPENDIX B

REENTRY TRAJECTORY EQUATIONS

The following simplified equations derived for flight within the atmosphere were used for the example problem herein. The primary assumptions include a spherical nonrotating earth, small flight-path angles, and a constant gravity term. The derivation of these equations and their applicability have been considered in a number of reports.¹¹

$$\ddot{h} = -g + \frac{V^2}{r} + \left(\frac{C_D A}{m} \right) \frac{1}{2} \rho V^2 \left(\frac{L}{D} - \frac{\dot{h}}{V} \right)$$

$$\dot{V} = - \left(\frac{C_D A}{m} \right) \frac{1}{2} \rho V^2$$

$$\psi = \int_{t_0}^{t_f} V dt$$

$$\varphi = 1.7 \times 10^{-8} \int_{t_0}^{t_f} \sqrt{\rho} V^3 dt$$

where

- $\frac{C_D A}{m}$ drag loading, 2.0 ft²/slug
- g local gravitational acceleration, 32.2 ft/sec²
- h altitude, ft
- $\frac{L}{D}$ control value of lift-drag ratio
- r radius from earth center, 21.1×10^6 ft
- V horizontal velocity, fps
- ρ atmosphere density, $0.00237 e^{-h/23,500}$ slug/ft³
- φ total heat input, Btu/ft²
- ψ final range, ft

APPENDIX C

DESCRIPTION OF COMPUTER SYSTEMS

In order to make meaningful a comparison of the results obtained from the analog and digital simulations, it is necessary to very briefly describe the computer systems used.

The analog computer was an EAI 231R-V equipped with electronic mode control of the amplifiers. The logic element of the hybrid simulation was an EAI DOS 350. The DOS 350 has a patchboard which permits one to combine logical elements, such as AND gates, flip-flops, shift registers, counters, etc., into complicated logic systems. It also has several delay line memories of various lengths as well as A-D and D-A converters for communicating between the DOS 350 and the analog computers.

The digital computer was an EAI 8400 mode 0 computer which had a $2\mu\text{sec}$ memory access time, an average floating point add time of approximately $13\mu\text{sec}$, an average floating point multiply time of approximately $15\mu\text{sec}$, and a floating point word size of 32 bits.

REFERENCES

1. G. Leitmann, ed., *Optimization Techniques*, Academic Press, 1962.
2. A. V. Balakrishnan and L. W. Neustadt, eds., *Computing Methods in Optimization Problems*, Academic Press, 1964.
3. R. C. Wingrove, et al, "A Method of Trajectory Optimization by Fast-Time Repetitive Computations," NASA TN D-3404, 1966.
4. A. E. Bryson and W. F. Denham, "A Steepest-Ascent Method for Solving Optimum Programming Problems," Raytheon Rep. BR 1303, 1961. Also *J. Appl. Mech.*, vol. 29, no. 2, (June 1962), pp. 247-257.
5. H. J. Kelley, "Gradient theory of optimal flight paths," *ARS J.*, vol. 30, no. 10, (Oct. 1960), pp. 947-954.
6. A. E. Bryson, et al, "Determination of lift or drag programs that minimize reentry heating," *J. Aerospace Sci.*, vol. 29, no. 4, (April 1962), pp. 420-430.
7. H. Blanton, ed., "Three-Dimensional Trajectory Optimization Study, Pt. 1—Optimum Programming Formulation," NASA CR-57030, 1964. (Supersedes Aero. Sys. Div. Rep. ASD-TDR-62-295 and Raytheon Rep. BR-1759-1)
8. R. W. Hamming, *Numerical Methods for Scientists and Engineers*, McGraw-Hill, 1962.
9. A. Ralston, *A First Course in Numerical Analysis*, McGraw-Hill, 1965.
10. J. H. Lundell, et al, "Experimental Investigation of a Charring Ablative Material Exposed to Combined Convective and Radiative Heating in Oxidizing and Nonoxidizing Environments," *Proc. AIAA Entry Technology Conf.*, Oct. 1964, pp. 216-227.
11. D. R. Chapman, "An Approximate Analytical Method for Studying Entry Into Planetary Atmospheres," NASA TR R-11, 1959.

1966 FALL JOINT COMPUTER CONFERENCE COMMITTEE

Chairman

R. GEORGE GLASER, McKinsey & Company, Inc.

Vice Chairman

LOUIS J. LAULER, Lockheed Missiles & Space Company

Secretary

JOHN F. MATTHEWS II, International Business Machines Corporation

Treasurer

JOHN B. SPRING, Price Waterhouse & Co.

Technical Program

WILLIAM H. DAVIDOW, Hewlett-Packard Company, Chairman

GLENN C. BACON, International Business Machines Corporation, Vice Chairman

LOUIS FELDNER, Lockheed Missiles & Space Company

MAURICE H. HALSTEAD, Lockheed Missiles & Space Company

JOHN R. HERNDON, Stanford Research Institute

GENE M. AMDAHL, International Business Machines Corporation

DONN B. PARKER, Control Data Corporation

WILLIAM D. CAMERON, NASA Ames Research Center

RICHARD C. REYNA, Hewlett-Packard Company

Exhibits

RAYMOND D. SMITH, SCM Corporation, Chairman

RICHARD DORRANCE, URS Corporation, Vice Chairman

Local Arrangements

THOMAS C. BIEG, International Business Machines Corporation, Chairman

RALPH R. WHEELER, Lockheed Missiles & Space Company, Vice Chairman

MARJORIE F. HILL, Control Data Corporation

HAROLD O. MARTIN, JR., Friden, Inc.

ROBERT WHYTE, International Business Machines Corporation

Registration

THOMAS R. DINES, Control Data Corporation, Chairman

DAVID KATCH, Control Data Corporation, Vice Chairman

Printing and Mailing

A. A. WICKS, Control Data Corporation, Chairman

JOHN M. GOWAN, Control Data Corporation, Vice Chairman

MARJORIE WEST, Control Data Corporation

VERYL J. MERRITT, Control Data Corporation

Publications

ALBERT C. PORTER, California Public Utilities Commission, Chairman

ROBERT L. BARRINGER, Arthur D. Little, Inc., Vice Chairman

MARY TESTA, Arthur D. Little, Inc.

Public Relations

MICHAEL J. MCCLUSKEY, International Business Machines Corporation, Chairman

JERRY M. KELLY, Memorex Corporation, Vice Chairman

DONALD FLETCHER, URS Corporation

PATRICK MURPHY, AMPEX Corporation

JOHN F. SCHEFFEL, International Business Machines Corporation

Ladies' Program

MRS. MARILYN DE C. RICHARDSON, International Business Machines Corporation, Chairman

MISS MARY A. STEWART, International Business Machines Corporation, Vice Chairman

MISS MARY ANN HOTOP, International Business Machines Corporation

MISS SARA TROY, International Business Machines Corporation

MISS EVANGELINE YOUNG, International Business Machines Corporation

Education Program

ROBERT J. ANDREWS, International Business Machines Corporation, Chairman

NORTON SALBERG, International Business Machines Corporation, Vice Chairman

ROBERT HASELTINE, Radio Corporation of America

Science Theater

THOMAS C. BORRELLI, Lockheed Missiles & Space Company

Consultant

WILLIAM C. ESTLER, Public Relations

Advisors

CHARLES P. BOURNE, Programming Services, Inc., ADI Advisor

DONN B. PARKER, Control Data Corporation, ACM Advisor

JOHN E. SHERMAN, Lockheed Missiles & Space Company, SCI Advisor

RICHARD I. TANAKA, California Computer Products, Inc., IEEE Advisor

JOHN H. WAHLGREN, University of California, AMTCL Advisor

REFEREES PANELISTS, AND SESSION CHAIRMEN

REFEREES

C. T. Abraham	R. Furman	A. W. Potts
M. Alston	A. E. Gardner	R. L. Pritchard
E. B. Altman	V. H. Grinich	A. Ralston
G. N. Arnovick	H. P. Hartkemeier	L. C. Ray
C. J. Bell	D. B. Hildebrand	E. K. Ritter
J. Bliss	E. L. Jacks	L. G. Roberts
G. Bolton	W. J. Karplus	J. Robinson
R. W. Borneman	R. W. Koepcke	C. B. Rosen
G. Bowlby	G. A. Korn	M. Rosenberg
J. R. Brown, Jr.	L. Kovach	A. Rubin
W. C. Carter	J. H. Kuney	T. R. Savage
S. H. Chasen	J. L. Lasser	R. J. Shook
T. E. Cheatham, Jr.	P. Lazarus	R. Silver
K. Chuang	D. C. Lincicome	R. Singleton
C. Clewlow	C. R. Lindholm	B. Smith
A. B. Clymer	J. T. Lundy	R. V. Smith
J. B. Cohen	M. E. McCoy	W. R. Smith
J. Cunningham	W. V. Mansfield	R. J. Spinrad
F. J. Damerau	R. L. Mattson	T. B. Steel
C. Dickens	C. H. Mays	T. Stockham, Jr.
R. Duda	E. E. Mitchell	R. Summit
E. D. Dwyer	M. Montalbano	W. P. Timlake
T. J. Dylewski	J. D. Murphy	J. F. Traub
T. S. Eason	I. D. Nehama	V. Weaver
R. Elfant	R. Norman	D. A. Williams
B. Elspas	T. W. Olle	W. H. Williams
F. Engel, Jr.	J. Parsons	P. Wilson
D. Euglabart	H. Petersen	N. Wirth
S. Estes	S. Petrick	J. Wolle
J. L. Fick	R. Pinkham	H. Wolpe
W. D. Frazer	N. E. Pobanz	J. W. Young, Jr.
J. Friedman	J. H. Pomerene	

PANELISTS

C. W. Adams	R. Howe	R. V. Smith
P. Armer	C. A. Jones	T. B. Steel
S. H. Chasen	D. Keehn	A. Taub
C. R. DeCarlo	J. C. R. Licklider	F. V. Wagner
D. Engelbart	H. F. Meissinger	W. H. Wattenburg
B. D. Fried	T. J. O'Rourke	J. W. Weil
H. Friedman	A. Rogers	N. Wirth
R. Glaser	J. A. G. Russell	J. Wolle
D. Hall	O. Selfridge	

SESSION CHAIRMEN

L. H. Amaya
G. M. Amdahl
J. B. Angell
G. H. Ball
P. Baran
W. Brunner
S. Brunner

S. Fernbach
H. R. J. Grosch
R. W. Hamming
H. D. Huskey
J. D. Kuehler
W. C. McGee
R. G. Mills
T. J. Moffett

W. R. Nugent
J. A. Rajchman
N. J. Ream
R. Rice
A. C. Soudack
R. Vichnevetsky
H. Von Foerster

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES (AFIPS)

211 E. 43rd Street, New York, New York 10617

Officers and Board of Directors

Chairman

DR. BRUCE GILCHRIST*
IBM Corporation
Data Processing Division
112 East Post Road
White Plains, New York 10601

Chairman-Elect

MR. PAUL ARMER*
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Secretary

MR. MAUGHAN S. MASON
Dept. 210
IBM Corporation—FSD
P. O. Box 1250
Huntsville, Alabama 35805

Treasurer

MR. WILLIAM D. ROWE*
Sylvania Electronics Systems
189 B. Street
Needham Heights, Massachusetts

ACM Directors

DR. ANTHONY G. OETTINGER
Harvard Computation Laboratory
Cambridge, Massachusetts 02138

DR. ROBERT W. RECTOR*
Informatics, Inc.
5430 Van Nuys Boulevard
Sherman Oaks, California 91401

MR. J. D. MADDEN
ACM Headquarters
211 East 43rd Street
New York, New York 10017

DR. WALTER HOFFMAN
Computing Center
Wayne State University
Detroit, Michigan 48202

IEEE Directors

MR. SAMUEL LEVINE
Bunker-Ramo Corporation
445 Fairfield Avenue
Stamford, Connecticut 06904

MR. KEITH W. UNCAPHER
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

DR. T. J. WILLIAMS
Control & Information Systems Laboratory
Purdue University
Lafayette, Indiana 47907

DR. R. I. TANAKA*
California Computer Products
305 North Muller Street
Anaheim, California 92803

Simulation Councils Director

MR. JOHN E. SHERMAN*
Lockheed Missiles & Space Co.
D-59-10, B-151
P. O. Box 504
Sunnyvale, California 94088

American Documentation Institute Director

MR. HAROLD BORKO
System Development Corporation
2500 Colorado Avenue
Santa Monica, California 90406

Association for Machine Translation and Computational Linguistics-Observer

DR. DAVID G. HAYS
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Executive Secretary

MR. H. G. ASMUS
AFIPS Headquarters
211 East 43rd Street
New York, New York 10017

* Executive Committee

AFIPS Committee Chairmen

Abstracting

DR. DAVID G. HAYS
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Admissions

MR. WALTER L. ANDERSON
General Kinetics Inc.
2611 Shirlington Road
Arlington, Virginia 22206

Awards

DR. ARNOLD A. COHEN
UNIVAC
2276 Highcrest Drive
Roseville, Minnesota 55113

Conference

DR. MORTON M. ASTRAHAN
IBM Corporation—ASDD
P. O. Box 66
Los Gatos, California 95030

Constitution and By-Laws

MR. MAUGHAN S. MASON
Dept. 210
IBM Corporation—FSD
P. O. Box 1250
Huntsville, Alabama 35805

Education

DR. MELVIN SHADER
IBM Corporation
100 Westchester Avenue
Harrison, New York 10528

Finance

MR. WALTER M. CARLSON
Director of Technical Information
Office of the Director of Defense
Research and Engineering
Washington, D. C. 20301

Harry Goode Memorial Award

DR. WILLIS H. WARE
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

IFIP Congress 68

DR. DONALD L. THOMSEN, JR.
IBM Corporation
Old Orchard Road
Armonk, New York 10504

International Relations

DR. EDWIN L. HARDER
1204 Milton Avenue
Pittsburgh, Pennsylvania 15218

Planning

DR. JACK MOSHMAN
CEIR, Inc.
EBS Management Consultants, Inc.
1625 I Street, N.W.
Washington, D. C. 20006

Public Relations

MR. ISAAC J. SELIGSOHN
IBM Corporation
Old Orchard Road
Armonk, New York 10504

Publications

MR. STANLEY ROGERS
P. O. Box R
Del Mar, California 92014

Publications Task Force

MR. SOL ROSENTHAL
HQ USAF
AFA DAC
Pentagon
Washington, D. C. 20330

*Social Implications of Information
Processing Technology*

MR. PAUL ARMER
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Technical Program

MR. JACK ROSEMAN
2313 Coleridge Drive
Silver Spring, Maryland 20910

Newsletter

MR. DONALD B. HOUGHTON, 15-W
Westinghouse Electric Corporation
3 Gateway Center, Box 2278
Pittsburgh, Pennsylvania 15230

COSATI Liaison

MR. GERHARD L. HOLLANDER
Hollander Associates
P. O. Box 2276
Fullerton, California 92633

Consultant

MR. HARLAN E. ANDERSON
Rollingwood Lane
Concord, Massachusetts 01742

JCC General Chairmen

1966 FJCC

MR. R. GEORGE GLASER
McKinsey & Company, Inc.
100 California Street
San Francisco, California 94111

1967 FJCC

MR. LINDER C. HOBBS
Hobbs Associates, Inc.
P. O. Box 686
Corona del Mar, California 92625

1967 SJCC

MR. BRIAN POLLARD
Product Planning
RCA-EDP
Building 204-2
Cherry Hill, New Jersey 08101

1966 FJCC LIST OF EXHIBITORS

Academic Press, Inc.
Adage, Inc.
Addison-Wesley Publishing Co., Inc.
Advanced Scientific Instruments Div., EMR, Inc.
Allen-Babcock Computing, Inc.
Amp, Inc.
Ampex Corporation
Anelex Corporation
Applied Data Research, Inc.
Applied Dynamics, Inc.
Auto-trol Corporation

Bell System, A.T.&T.
Benson-Lehner Corporation
Bryant Computer Products Div., Ex-Cell-O Corp.
Burroughs Corporation, Electronic Components Div.
Business Information Technology

California Computer Products, Inc.
Calma Company
Comcor, Inc.
Computer Accessories Corporation
Computer Design Publishing Corporation
Computer Products, Inc.
Computer Sciences Corporation
Computers and Automation
Conrac Division of Giannini Controls Corporation
Consolidated Electrodynamics Corporation
Control Data Corporation
Corning Glass Works
Cybetronics, Inc.

Data Equipment Company, A Div. of BBN Inc.
Data Machines, Inc.
Decision Control, Inc.
Data Pathing, Inc.
Datamation
Data Processing Digest, Inc.
Data Processing Magazine
Data Products Corporation
DI/AN Controls, Inc.
Digital Equipment Corporation
Digitek Corporation
Digitronics Corp.

Eastman Kodak Company, Business Systems Markets
Div.
E-H Research Laboratories, Inc.
Elco Corporation

Electronic Associates, Inc.
Electronic Memories, Inc.

Fabri-Tek Incorporated
Fairchild Semiconductor
Ferroxcube Corporation
Friden, Inc.

General Computers, Inc.
General Dynamics Corp., Electronics Div.
General Electric Information Systems Div.
General Kinetics Incorporated
General Precision/Librascope Group
Geo Space Corporation
The Gerber Scientific Instrument Co.

Hewlett-Packard Datamec Division
Hewlett-Packard Dymec Division
Honeywell, Inc., Computer Control Div.

Indiana General Corporation
Information Displays, Inc.
International Business Machines Corp., DP Div.
ITT Data Services
ITT-Industrial Products Div.

Kennedy Company
Kleinschmidt, Div. of SCM Corp.

Lear Siegler, Inc., Data & Controls Div.
Lockheed Electronics Company

Magne-Head, A Div. of General Instrument Corp.
MAI Equipment Corporation
McGraw-Hill Book Co.
Memorex Corporation
Midwestern Instruments, Inc.
Milgo Electronic Corporation
3M, Revere-Mincom Div.
Monroe DATALOG Division of Litton Industries

The National Cash Register Co.
Navigation Computer Corporation
North Atlantic Industries, Inc.

Photocircuits Corporation
Potter Instrument Company, Inc.
Precision Instrument Company
Prentice Hall, Inc.

RCA, E.C.&D. Div.
RCA, EDP Div.
Raytheon Computer
Rheem Electronics
Rixon Electronics, Inc.
Rotron Manufacturing Co., Inc.
Roytron Div., Royal Typewriter Co., Inc.

Sanders Associates, Inc.
Scientific Control Systems, Inc.
Scientific Data Systems, Inc.
Simulators, Inc.
Soroban Engineering, Inc.
Spartan Books
Stromberg-Carlson Corp., Data Products Div.
Systron-Donner Corporation

Tally Corporation
Tasker Instruments Corporation
Teletype Corporation

Texas Instruments Inc., Semiconductor-Components
Div.
Texas Instruments Inc., Apparatus Div.
Thin Film, Inc.
Thompson Book Company
Toko N. Y. Inc.
Transistor Electronics Corporation
Tymshare, Inc.

Ultronic Systems Corporation
UNIVAC Division of Sperry Rand Corporation
Uptime Corporation
URS Corporation
U.S. Magnetic Tape Co.

Vermont Research Corporation

Western Union Telegraph Co.
John Wiley & Sons, Inc.
Wyle Laboratories

Zeltex, Inc.

AUTHOR INDEX

- ABRAHAMS, P., 661
 AMIOT, L. W., 743, 759
 BARNETT, JEFFREY A., 661
 BECKER, C. H., 711
 BEKEY, G. A., 191
 BEREZNER, S. C., 365
 BITZER, D. L., 541
 BLUMENTHAL, S. C., 579
 BOOK, ERWIN, 661
 BRACKETT, J., 465
 BRYANT, L. T., 743, 759
 BURGER, J. F., 357
 BUSCH, K. J., 445
 CARNEY, H. C., 365
 CARTER, J. C., 743
 CATT, I., 315
 CHEATHAM, T. E., JR., 623
 CHONG, C. F., 305
 CRAIG, J. A., 365
 CRAN, M. H., 191
 CRANDALL, R. L., 1
 CRAWFORD, B. J., 247
 DANTINE, D. J., 303
 DARLEY, D. L., 37
 DEANDRADE, A. B., 247
 DERTOUZOS, M. I., 201
 DODD, G. G., 677
 ECKHART, B. J., 23
 ELLINGER, P. B., 293
 ENGLAND, A. W., 51
 EVANS, T. G., 37
 FARR, L., 513
 FIRTH, DONNA, 661
 FLYNN, M. J., 97
 FORGIE, J. W., 433
 GARTH, E. C., 315
 GENTLEMEN, W. M., 563
 GOSDEN, J. A., 651
 GRAHAM, H. L., 201
 GRONER, G. F., 591
 HAKOLA, V. F., 579
 HALPERN, M., 639
 HAWKINSON, LOWELL, 661
 HIGGINS, G. C., 729
 HITTEL, L. A., 395
 HOBBS, L. C., 89
 HOFFMAN, R. B., 523
 HOOVER, E. S., 23
 HUBACHER, E. H., 229
 ISAACS, H. H., 505
 KAMENY, STANLEY L., 661
 KAPLOW, R., 465
 KARSTAD, K., 333
 KAUFMAN, B. A., 293
 KENNEDY, J. R., 211
 KERBY, H. R., 735
 KETCHPEL, R. D., 531
 KOFORD, J. S., 229
 KUEHLER, J. D., 735
 KUNEY, J. H., 149
 KUNKEL, G. Z., 157
 KUNO, H. J., 293
 LAMBERTS, R. L., 729
 LAZORCHAK, B. G., 149
 LEHRER, N. H., 531
 LEVIN, MICHAEL I., 661
 LITTLE, W. D., 181
 LONG, R. E., 357
 LONGYEAR, C. R., 365
 MAKRIS, C. J., 137
 MARILL, T., 425
 MARSAGLIA, G., 169
 MCCABE, L. B., 513
 MCCALLISTER, J. P., 305
 MCMAHON, M. T., 247
 MEADOW, C. T., 381
 MEDDAUGH, S. A., 281
 MEDWEDEFF, M., 1
 MENDELSON, M. J., 51
 MIESSNER, W. W., 789
 MOORE, D. W., 267
 MURRAY, D. E., 315
 NEBEL, B. E., 115
 NEWBERRY, S. P., 717
 NOYCE, R., 111
 ONYSHKEVYCH, L., 333
 PEARSON, K. L., 281
 PERRY, J. H., JR., 125
 PETRITZ, R. L., 65
 PRATT, T. W., 613
 PRICE, D. G., 501
 RABY, J. S., 799
 ROBERTS, L. G., 223, 425
 RUBIN, A. E., 771
 RYAN, J. L., 1
 SABROFF, A. E., 191
 SANATHANAN, C. K., 743
 SANDE, G., 563
 SANDER, W. B., 105
 SAUNDERS, ROBERT A., 661
 SEBESTYEN, G., 685
 SHAHBENDER, R., 333
 SHEPPS, L., 771
 SHERMAN, D., 149
 SIMMONS, R. F., 357
 SLOTTOW, H. G., 541
 SPORZYNSKI, G. A., 229
 STALLINGS, C. B., 549
 STAMBLER, L., 413
 STEIN, R. P., 759
 STICKNEY, G. F., 479
 STOWE, A. N., 433
 STRAUSS, J. C., 603
 STRICKER, J. L., 789
 STRICKLAND, P. R., 229
 STRONG, S., 465
 THOMPSON, F. B., 349
 TUKEY, J., 695
 VIDAL, J. J., 175
 WALCAVICH, S. W., 149
 WAUGH, D. W., 381
 WAXMAN, R., 247
 WEINSTEIN, H., 333
 WEISSMAN, CLARK, 661
 WIESEN, R. A., 433
 WILK, M., 695
 WINGROVE, R. C., 799
 WONG, A., 191
 YNTEMA, D. B., 433

Conferences 1 to 19 were sponsored by the National Joint Computer Committee, predecessor of AFIPS. Conferences 20 and up are sponsored by AFIPS. Copies of volumes 1-28, Part II may be purchased from SPARTAN BOOKS, Scientific and Technical Division of Books, Inc., 432 Park Avenue South, New York, N. Y.

<i>Volume</i>	<i>Part</i>	<i>List Price</i>	<i>Member Price</i>
1-3		11.00	11.00
4-6		9.00	9.00
7-9		9.00	9.00
10, 11		7.00	7.00
12, 13		7.00	7.00
14, 15		8.00	8.00
16, 17		6.00	6.00
18		3.00	3.00
19		3.00	3.00
20		12.00	12.00
21		6.00	6.00
22		8.00	4.00
23		10.00	5.00
24		16.50	8.25
25		16.00	8.00
26	I	18.75	9.50
26	II	4.75	2.50
27		28.00	14.00
28		12.15	6.10

Cumulative Index to Vols. 1-26, Part II \$3.00

Vol. 29. 1966 Fall Joint Computer Conference,
San Francisco, California, 1966